Explain the linear regression algorithm in detail.

Regression analysis is a form of predictive modeling technique which investigates the relationship between dependent/target variable with the independent/predictor variable.

Types of Regression:
Linear Regression
Logistic Regression
Polynomial Regression
Stepwise Regression

Linear Regression is a supervised machine learning algorithm, where the output to be predicted is a continuous variable and has a constant slope. Used to predict the values within a continuous range(e.g. sales, price).Linear Regression predicts the value of y based on the independent/predictor variable X. It finds the relationship between X and y and hence y and hence the name. Basically used for prediction and finding the relationship between the independent variable (x) with dependent variable (y). Linear regression is also known as *multiple regression*, *multivariate regression*, *ordinary least squares (OLS)*, and *regression.*

Two types in linear regression:

1) Simple Linear Regression: is a type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable.
2) The line for a simple linear regression model can be written as:
   y = b0 + b1 * x
   Where b0 and b1 are the coefficients we must estimate from the training data.

To understand Linear regression in a better way let's understand Cost function and Gradient descent.

Cost Function: provides the best fit line for the data points by helping us find the best values for b0 & b1. We need to minimize the error between the actual and the predicted values as we want the error to be least. We square the error difference and sum over all data points and divide that value by total number of data points. Therefore this cost function is called as Mean Squared Error (MSE). Using the cost function we are going to change the values of b0 & b1 such that the MSE values settle at minima.

Gradient Descent: is a method of updating b0 & b1 values such that MSE is minimum. The idea is to start with some value of b0 & b1 and then change the values of b0 & b1 iteratively to reduce the cost.

Linear Regression is a machine learning algorithm based on supervised learning.It performs a regression task to compute the regression coefficients.Regression models a target prediction based on independent variables.

Linear Regression performs the task to predict a dependent variable value (y) based on a given independent variable (x).So this regression technique finds out a linear relationship between x(input) and y(output).Hence it has got the name Linear Regression.The linear equation for univariate linear regression is given below

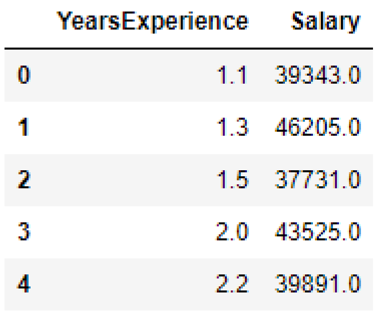
$$y = \theta_1 + \theta_2 . x$$

Equation of a Simple Linear Regression

y-output/target/dependent variable; x-input/feature/independent variable and theta1,theta2 are intercept and slope of the best fit line respectively, also known as regression coefficients.

Example Of A Simple Linear Regression:

Let us consider a example wherein we are predicting the salary a person given the years of experience he/she has in a particular field.The data set is shown below

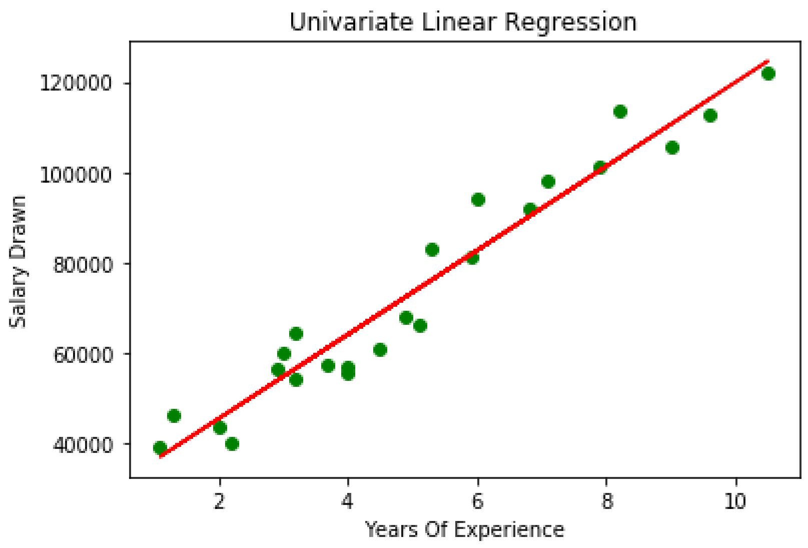


	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

Here x is the years of experience (input/independent variable) and y is the salary drawn (output/dependent/variable).

We have fitted a simple linear regression model to the data after splitting the data set into train and test.The python code used to fit the data to the Linear regression algorithm is shown below

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
plt.scatter(x_train,y_train,color='g')
plt.plot(x_train,lr.predict(x_train),color='r')
plt.xlabel("Years Of Experience")
plt.ylabel("Salary Drawn")
plt.title("Univariate Linear Regression")
plt.show()
```



The green dots represents the distribution the data set and the red line is the best fit line which can be drawn with theta1=26780.09 and theta2 =9312.57.
Note-theta1 is nothing but the intercept of the line and theta2 is the slope of the line. Best fit line is a line which best fits the data which can be used for prediction.

We'll build a linear regression model to predict `Sales` using an appropriate predictor variable.

Step 1: Data reading and Understanding

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
# Read the given CSV file, and view some sample records

advertising = pd.read_csv("advertising.csv")
advertising.head()
```

Let's inspect the various aspects of our dataframe

```
advertising.shape
```

```
(200, 4)
```

```
advertising.info()
```

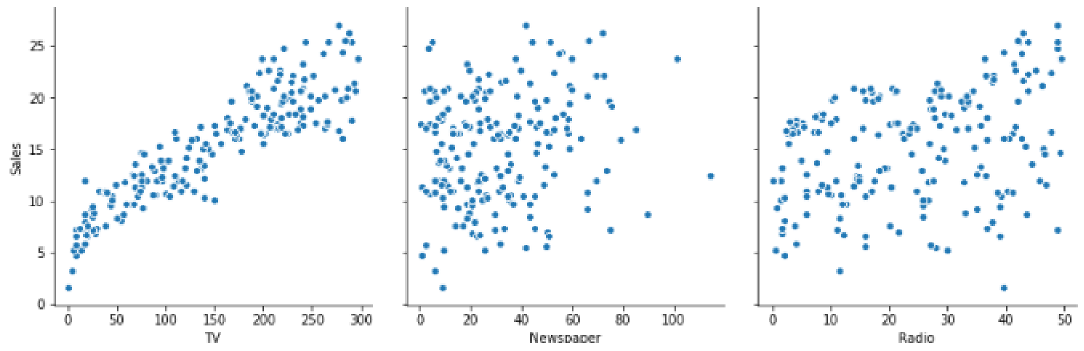
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
TV           200 non-null float64
Radio        200 non-null float64
Newspaper    200 non-null float64
Sales        200 non-null float64
dtypes: float64(4)
memory usage: 6.3 KB
```

```
advertising.describe()
```

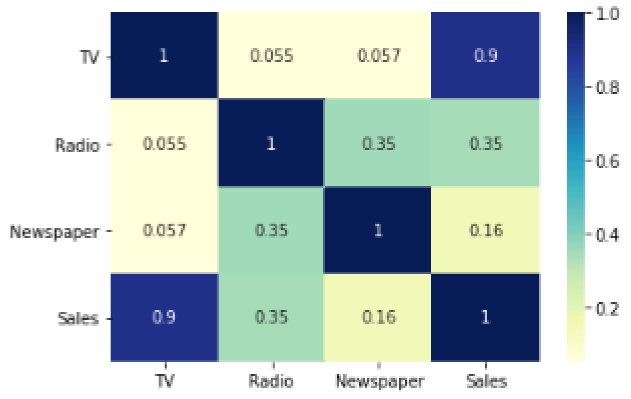
	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	15.130500
std	85.854236	14.846809	21.778621	5.283892
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	11.000000
50%	149.750000	22.900000	25.750000	16.000000
75%	218.825000	36.525000	45.100000	19.050000
max	296.400000	49.600000	114.000000	27.000000

Step 2: Visualize the data

```
sns.pairplot(advertising, x_vars=['TV', 'Newspaper', 'Radio'], y_vars='Sales',size=4, aspect=1, kind='scatter')
plt.show()
```



```python
sns.heatmap(advertising.corr(), cmap="YlGnBu", annot = True)
plt.show()
```



Step 3: Performing Simple Linear Regression

Generic Steps in model building using `statsmodels`

We first assign the feature variable, `TV`, in this case, to the variable `X` and the response variable, `Sales`, to the variable `y`.

```python
X = advertising['TV']
y = advertising['Sales']
```

Train-Test Split

You now need to split our variable into training and testing sets. You'll perform this by importing `train_test_split` from the `sklearn.model_selection` library. It is usually a good practice to keep 70% of the data in your train dataset and the rest 30% in your test dataset

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, test_size = 0.3, random_state = 100)
```

```python
# Let's now take a look at the train dataset

X_train.head()
```

```
74     213.4
3      151.5
185    205.0
26     142.9
90     134.3
Name: TV, dtype: float64
```

```python
y_train.head()
```

```
74     17.0
3      16.5
185    22.6
26     15.0
90     14.0
Name: Sales, dtype: float64
```

Building a Linear Model

You first need to import the `statsmodel.api` library using which you'll perform the linear regression.

```
import statsmodels.api as sm
```

By default, the `statsmodels` library fits a line on the dataset which passes through the origin. But in order to have an intercept, you need to manually use the `add_constant` attribute of `statsmodels`. And once you've added the constant to your `X_train` dataset, you can go ahead and fit a regression line using the `OLS` (Ordinary Least Squares) attribute of `statsmodels` as shown below

```
# Add a constant to get an intercept
X_train_sm = sm.add_constant(X_train)

# Fit the resgression line using 'OLS'
lr = sm.OLS(y_train, X_train_sm).fit()
```

```
# Print the parameters, i.e. the intercept and the slope of the regression line fitted
lr.params
```

```
const     6.948683
TV        0.054546
dtype: float64
```

\

```
# Performing a summary operation lists out all the different parameters of the regression line fitted
print(lr.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.816
Model:                            OLS   Adj. R-squared:                  0.814
Method:                 Least Squares   F-statistic:                     611.2
Date:                Thu, 13 Sep 2018   Prob (F-statistic):           1.52e-52
Time:                        22:39:43   Log-Likelihood:                -321.12
No. Observations:                 140   AIC:                             646.2
Df Residuals:                     138   BIC:                             652.1
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          6.9487      0.385     18.068      0.000       6.188       7.709
TV             0.0545      0.002     24.722      0.000       0.050       0.059
==============================================================================
Omnibus:                        0.027   Durbin-Watson:                   2.196
Prob(Omnibus):                  0.987   Jarque-Bera (JB):                0.150
Skew:                          -0.006   Prob(JB):                        0.928
Kurtosis:                       2.840   Cond. No.                         328.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

1. The coefficient for TV is 0.054, with a very low p value

The coefficient is statistically significant. So the association is not purely by chance.

2. R - squared is 0.816

Meaning that 81.6% of the variance in `Sales` is explained by `TV`

This is a decent R-squared value.

3. F statistic has a very low p value (practically low)

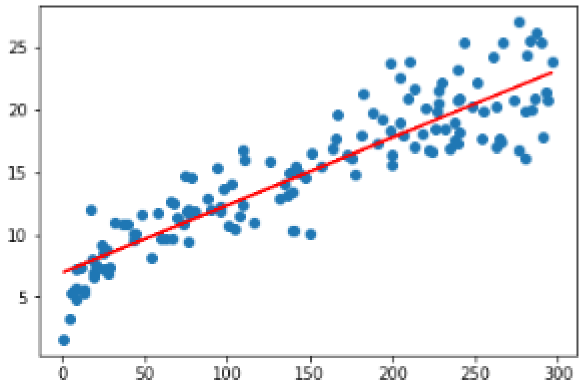
Meaning that the model fit is statistically significant, and the explained variance isn't purely by chance.

The fit is significant. Let's visualize how well the model fit the data.

From the parameters that we get, our linear regression equation becomes:

$$Sales = 6.948 + 0.054 \times TV$$

```python
plt.scatter(X_train, y_train)
plt.plot(X_train, 6.948 + 0.054*X_train, 'r')
plt.show()
```



Step 4: Residual analysis

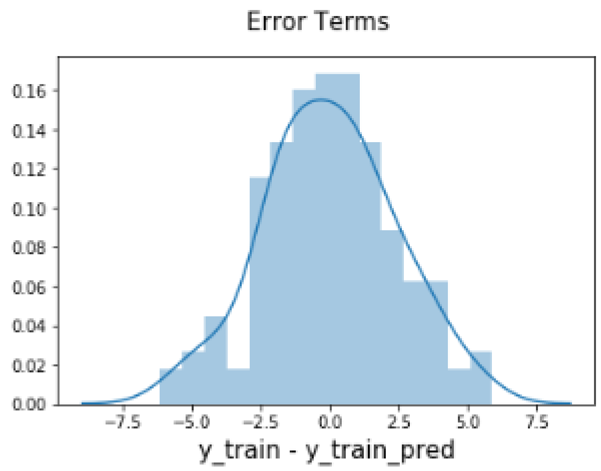
To validate assumptions of the model, and hence the reliability for inference

Distribution of the error terms

We need to check if the error terms are also normally distributed (which is infact, one of the major assumptions of linear regression), let us plot the histogram of the error terms and see what it looks like.

```python
y_train_pred = lr.predict(X_train_sm)
res = (y_train - y_train_pred)
```

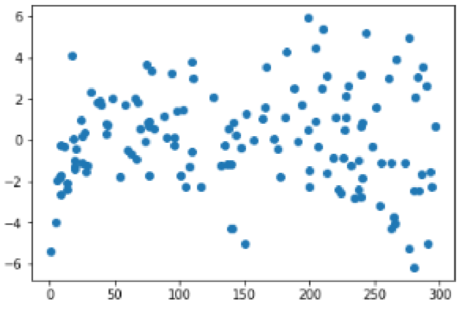
```python
fig = plt.figure()
sns.distplot(res, bins = 15)
fig.suptitle('Error Terms', fontsize = 15)                  # Plot heading
plt.xlabel('y_train - y_train_pred', fontsize = 15)         # X-label
plt.show()
```



The residuals are following the normally distributed with a mean 0. All good!

Looking for patterns in the residuals

```
plt.scatter(X_train,res)
plt.show()
```



We are confident that the model fit isn't by chance, and has decent predictive power. The normality of residual terms allows some inference on the coefficients.

Although, the variance of residuals increasing with X indicates that there is significant variation that this model is unable to explain.

As you can see, the regression line is a pretty good fit to the data

Step 5: Predictions on the Test Set

Now that you have fitted a regression line on your train dataset, it's time to make some predictions on the test data. For this, you first need to add a constant to the `X_test` data like you did for `X_train` and then you can simply go on and predict the y values corresponding to `X_test` using the `predict` attribute of the fitted regression line.

```
# Add a constant to X_test
X_test_sm = sm.add_constant(X_test)

# Predict the y values corresponding to X_test_sm
y_pred = lr.predict(X_test_sm)
```

```
y_pred.head()
```

```
126     7.374140
104    19.941482
99     14.323269
92     18.823294
111    20.132392
dtype: float64
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

Looking at the RMSE

```
#Returns the mean squared error; we'll take a square root
np.sqrt(mean_squared_error(y_test, y_pred))
```

```
2.019296008966233
```

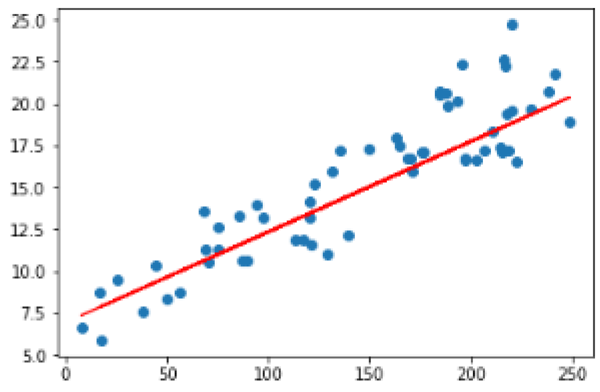
Checking the R-squared on the test set

```
r_squared = r2_score(y_test, y_pred)
r_squared
```

```
0.7921031601245658
```

Visualizing the fit on the test set

```
plt.scatter(X_test, y_test)
plt.plot(X_test, 6.948 + 0.054 * X_test, 'r')
plt.show()
```



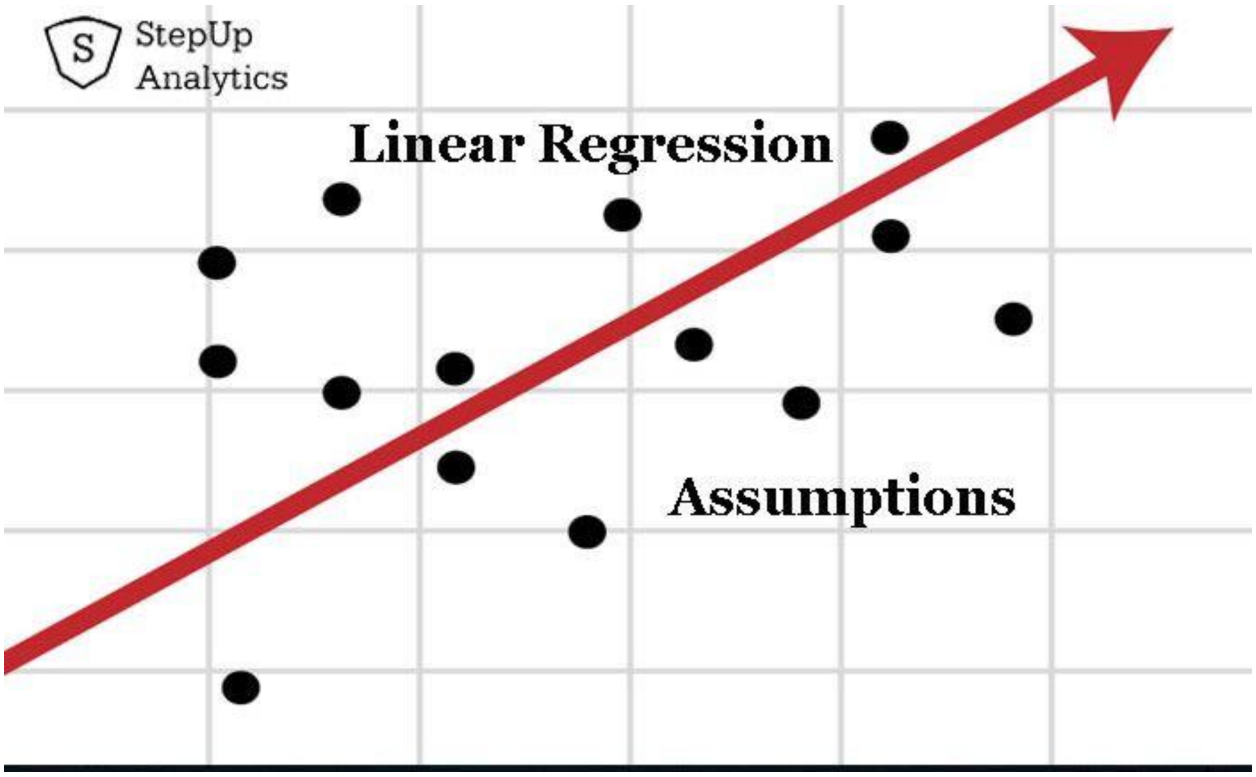
The equation we get is the same as what we got before!
$Sales=6.948+0.054*TV$Sales=6.948+0.054∗TV

Statsmodel linear model is useful as it is compatible with a lot of statsmodel utilites (cross validation, grid search etc.)

The same holds good for multiple linear regression where the only change is there will be multiple independent variable instead of one.

What are the assumptions of linear regression regarding residuals?

These assumptions which when satisfied while building a linear regression model produces a best fit model for the given set of data.



Linear Regression is a machine learning algorithm based on supervised learning.It performs a regression task to compute the regression coefficients.Regression models a target prediction based on independent variables.
Linear Regression performs the task to predict a dependent variable value (y) based on a given independent variable (x).So this regression technique finds out a linear relationship between x(input) and y(output).Hence it has got the name Linear Regression.The linear equation for univariate linear regression is given below

$$y = \theta_1 + \theta_2.x$$

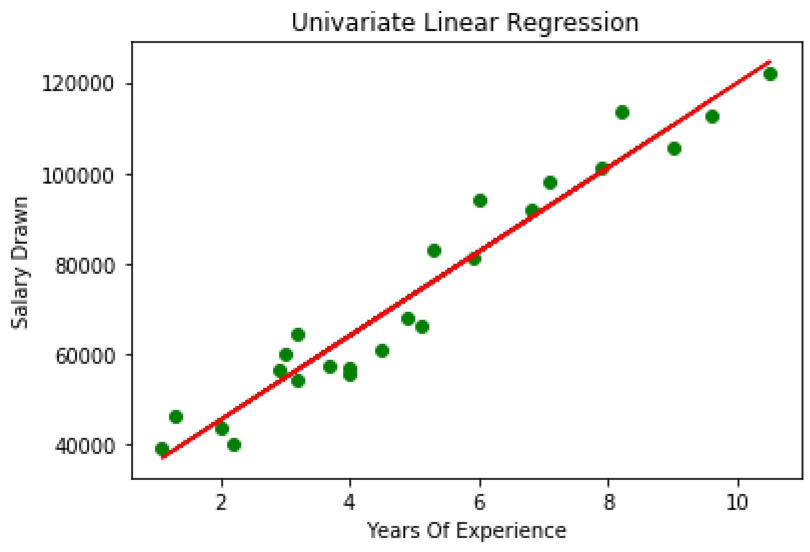
Equation of a Simple Linear Regression

y-output/target/dependent variable; x-input/feature/independent variable and theta1,theta2 are intercept and slope of the best fit line respectively, also known as regression coefficients.
Example Of A Simple Linear Regression:
Let us consider a example wherein we are predicting the salary a person given the years of experience he/she has in a particular field.The data set is shown below

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

Here x is the years of experience (input/independent variable) and y is the salary drawn (output/dependent/variable).
We have fitted a simple linear regression model to the data after splitting the data set into train and test.The python code used to fit the data to the Linear regression algorithm is shown below

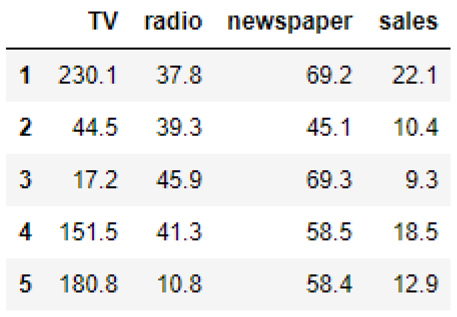
```python
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
plt.scatter(x_train,y_train,color='g')
plt.plot(x_train,lr.predict(x_train),color='r')
plt.xlabel("Years Of Experience")
plt.ylabel("Salary Drawn")
plt.title("Univariate Linear Regression")
plt.show()
```



The green dots represents the distribution the data set and the red line is the best fit line which can be drawn with theta1=26780.09 and theta2 =9312.57.
Note-theta1 is nothing but the intercept of the line and theta2 is the slope of the line.Best fit line is a line which best fits the data which can be used for prediction.

Description of the Data Set which is used for explaining the assumptions of linear regression

The data set which is used is the Advertising data set. This data set contains information about money spent on advertisement and their generated Sales. Money was spent on TV, radio and newspaper ads.It has 3 features namely TV, radio and newspaper and 1 target Sales.



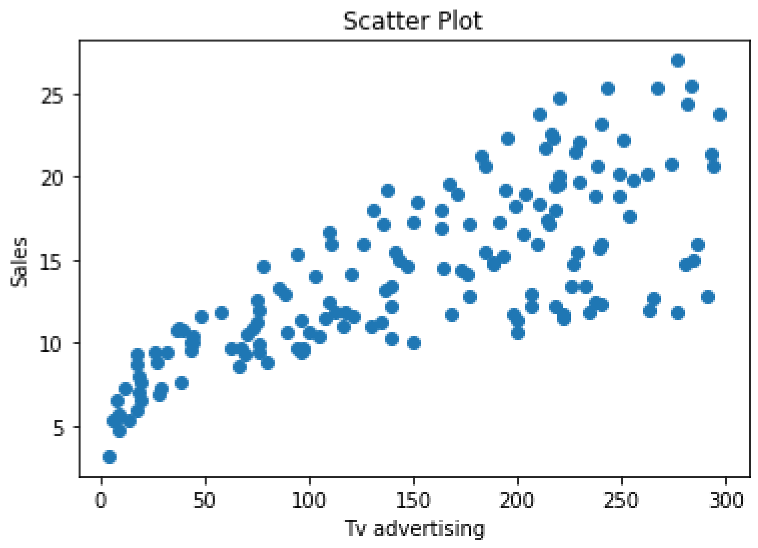
	TV	radio	newspaper	sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9

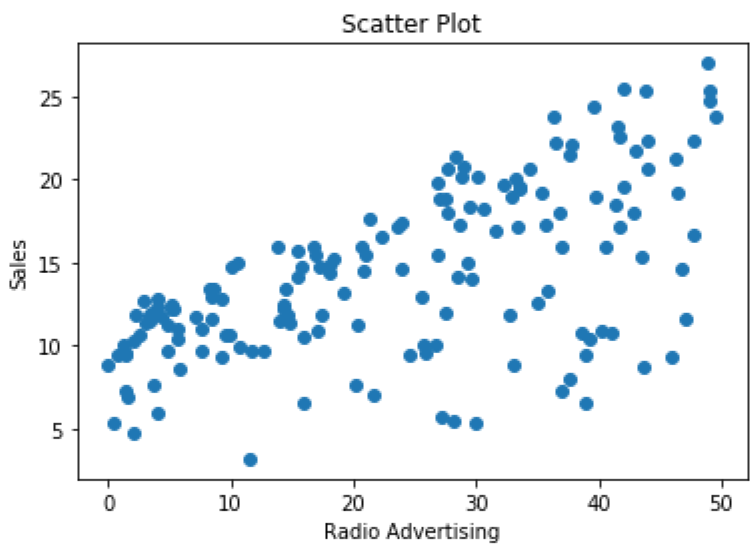
Assumptions of Linear Regression

There are 5 basic assumptions of Linear Regression Algorithm:

1. **Linear Relationship between the features and target:**

According to this assumption there is linear relationship between the features and target. Linear regression captures only linear relationship. This can be validated by plotting a scatter plot between the features and the target.



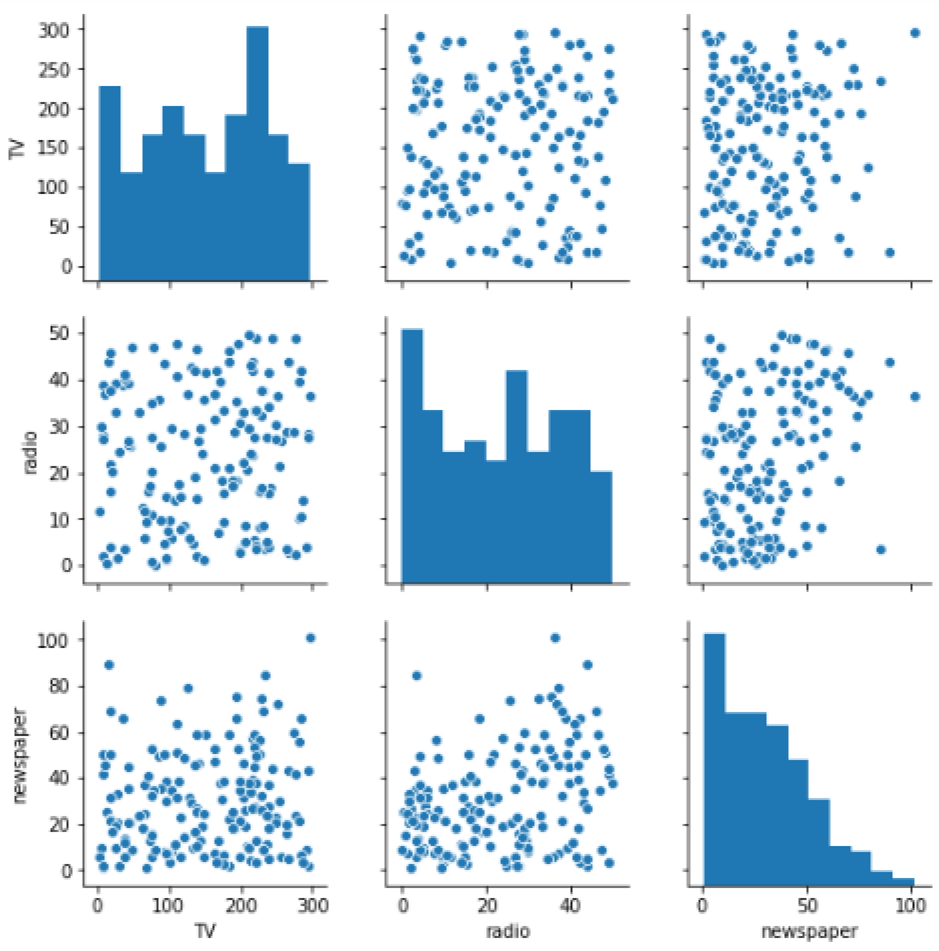


The first scatter plot of the feature TV v/s Sales tells us that as the money invested on TV advertisement increases the sales also increases linearly and the second scatter plot which is the feature Radio v/s Sales also shows a partial linear relationship between them, although not completely linear.

2. **Little or no Multicollinearity between the features:**

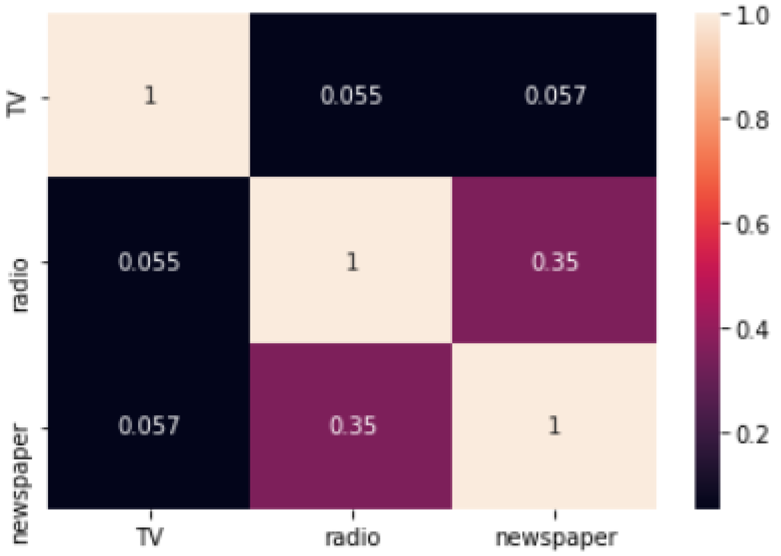
Multicollinearity is a state of very high inter-correlations or inter-associations among the independent variables. It is therefore a type of disturbance in the data if present weakens the statistical power of the regression model. Pair plots and heat maps (correlation matrix) can be used for identifying highly correlated features.
Multicollinearity may be tested with three central criteria:

1) Correlation matrix – when computing the matrix of Pearson's Bivariate Correlation among all independent variables the correlation coefficients need to be smaller than 1.
2) Tolerance – the tolerance measures the influence of one independent variable on all other independent variables; the tolerance is calculated with an initial linear regression analysis. Tolerance is defined as $T = 1 – R^2$ for these first step regression analysis. With $T < 0.1$ there might be Multicollinearity in the data and with $T < 0.01$ there certainly is.
3) Variance Inflation Factor (VIF) – the variance inflation factor of the linear regression is defined as $VIF = 1/T$. With $VIF > 5$ there is an indication that multicollinearity may be present; with $VIF > 10$ there is certainly multicollinearity among the variables



```
sns.heatmap(df_new.corr(),annot=True)
```

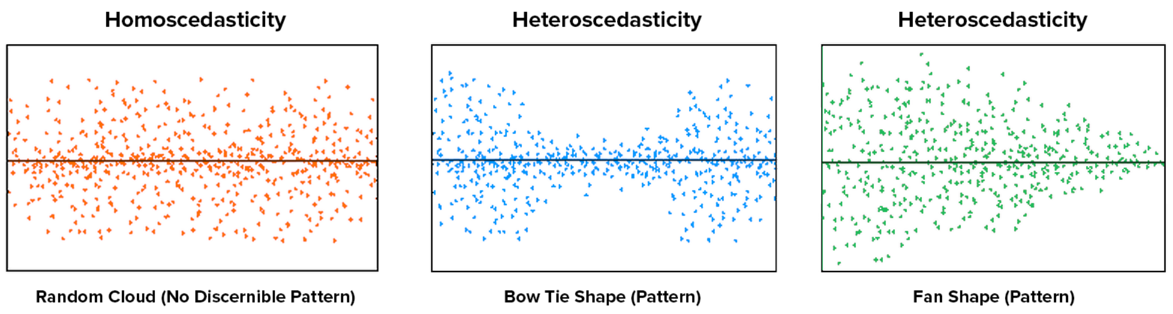
```
<matplotlib.axes._subplots.AxesSubplot at 0x1676c
```



This heat map gives us the correlation coefficients of each feature with respect to one another which are in turn less than 0.4.Thus the features aren't highly correlated with each other.
Why removing highly correlated features is important?
The interpretation of a regression coefficient is that it represents the mean change in the target for each unit change in an feature when you hold all of the other features constant. However, when features are correlated, changes in one feature in turn shifts another feature/features. The stronger the correlation, the more difficult it is to change one feature without changing another. It becomes difficult for the model to estimate the relationship between each feature and the target independently because the features tend to change in unison.
How multicollinearity can be treated?
If we have 2 features which are highly correlated we can drop one feature or combine the 2 features to form a new feature, which can further be used for prediction.

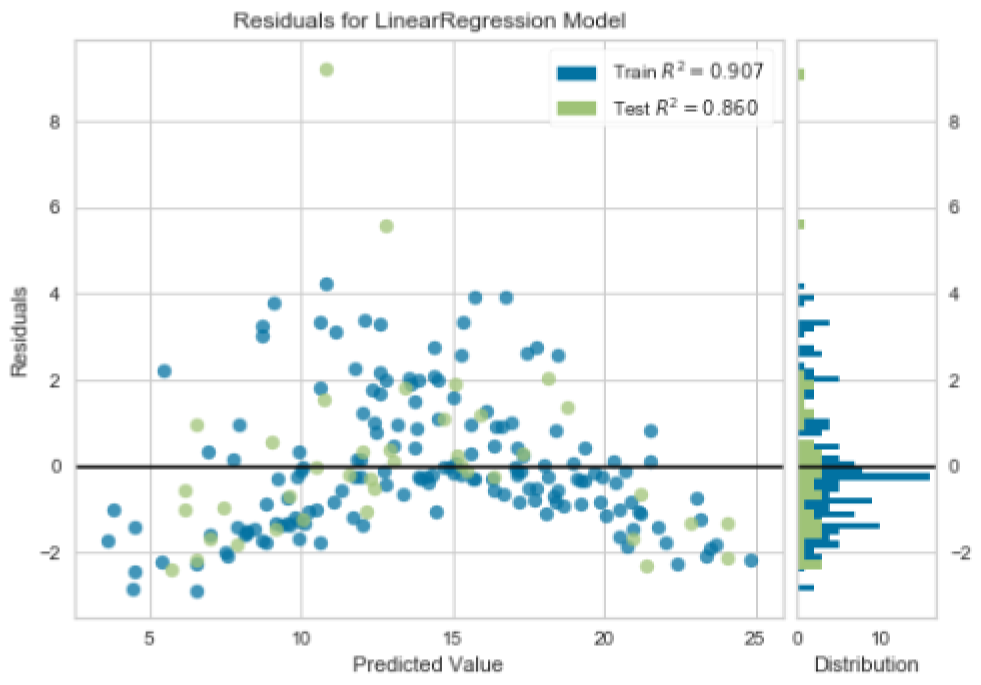
3. **Homoscedasticity Assumption:**

Homoscedasticity describes a situation in which the error term (that is, the "noise" or random disturbance in the relationship between the features and the target) is the same across all values of the independent variables. A scatter plot of residual values vs predicted values is a goodway to check for homoscedasticity. There should be no clear pattern in the distribution and if there is a specific pattern, the data is heteroscedastic.



The leftmost graph shows no definite pattern i.e constant variance among the residuals, the middle graph shows a specific pattern where the error increases and then decreases with the predicted values violating the constant variance rule and the rightmost graph also exhibits a specific pattern where the error decreases with the predicted values depicting heteroscedasticity
Python code for residual plot for the given data set:

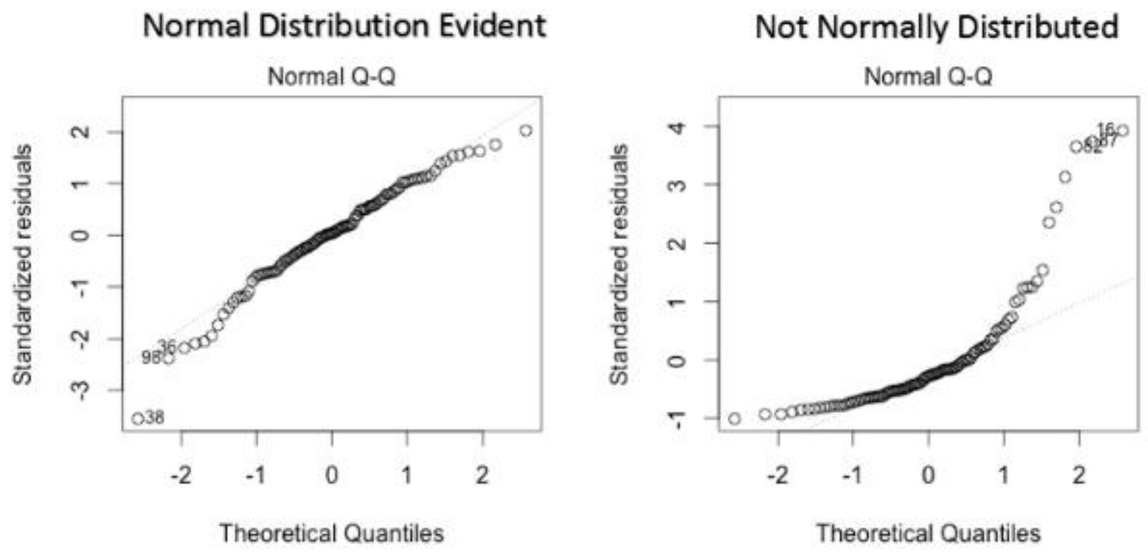
```
from yellowbrick.regressor import ResidualsPlot
from sklearn.linear_model import LinearRegression
Lr=LinearRegression()
visualizer = ResidualsPlot(Lr)
visualizer.fit(x_train,y_train)      # Fit the training data to the model
visualizer.score(x_test,y_test)      # Evaluate the model on the test data
visualizer.poof() visualizer.poof() # Draw/show/poof the data
```



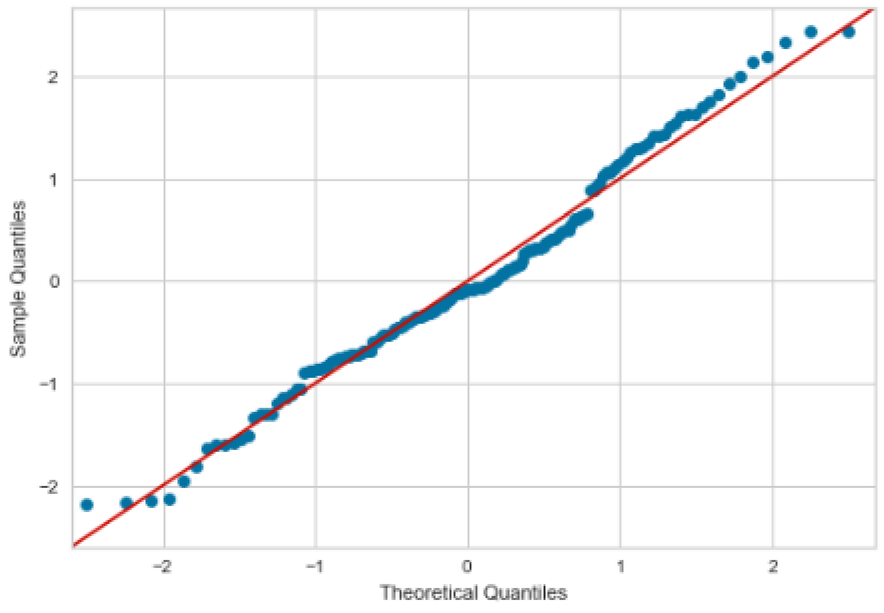
4. Normal distribution of error terms:

The fourth assumption is that the error (residuals) follows a normal distribution. However, a less widely known fact is that, as sample sizes increase, the normality assumption for the residuals is not needed. More precisely, if we consider repeated sampling from our population, for large sample sizes, the distribution (across repeated samples) of the ordinary least squares estimates of the regression coefficients follow a normal distribution. As a consequence, for moderate to large sample sizes, non-normality of residuals should not adversely affect the usual inferential procedures. This result is a consequence of an extremely important result in statistics, known as the central limit theorem.

Normal distribution of the residuals can be validated by plotting a q-q plot.



```
import statsmodels.api as sm
mod_fit = sm.OLS(y_train,x_train).fit()
res = mod_fit.resid # residuals
fig = sm.qqplot(res,fit=True,line='45')
plt.show()
```



Using the q-q plot we can infer if the data comes from a normal distribution. If yes, the plot would show fairly straight line. Absence of normality in the errors can be seen with deviation in the straight line.

The q-q plot of the advertising data set shows that the errors(residuals) are fairly normally distributed. The histogram plot in the “Error(residuals) vs Predicted values” in assumption no.3 also shows that the errors are normally distributed with mean close to 0.

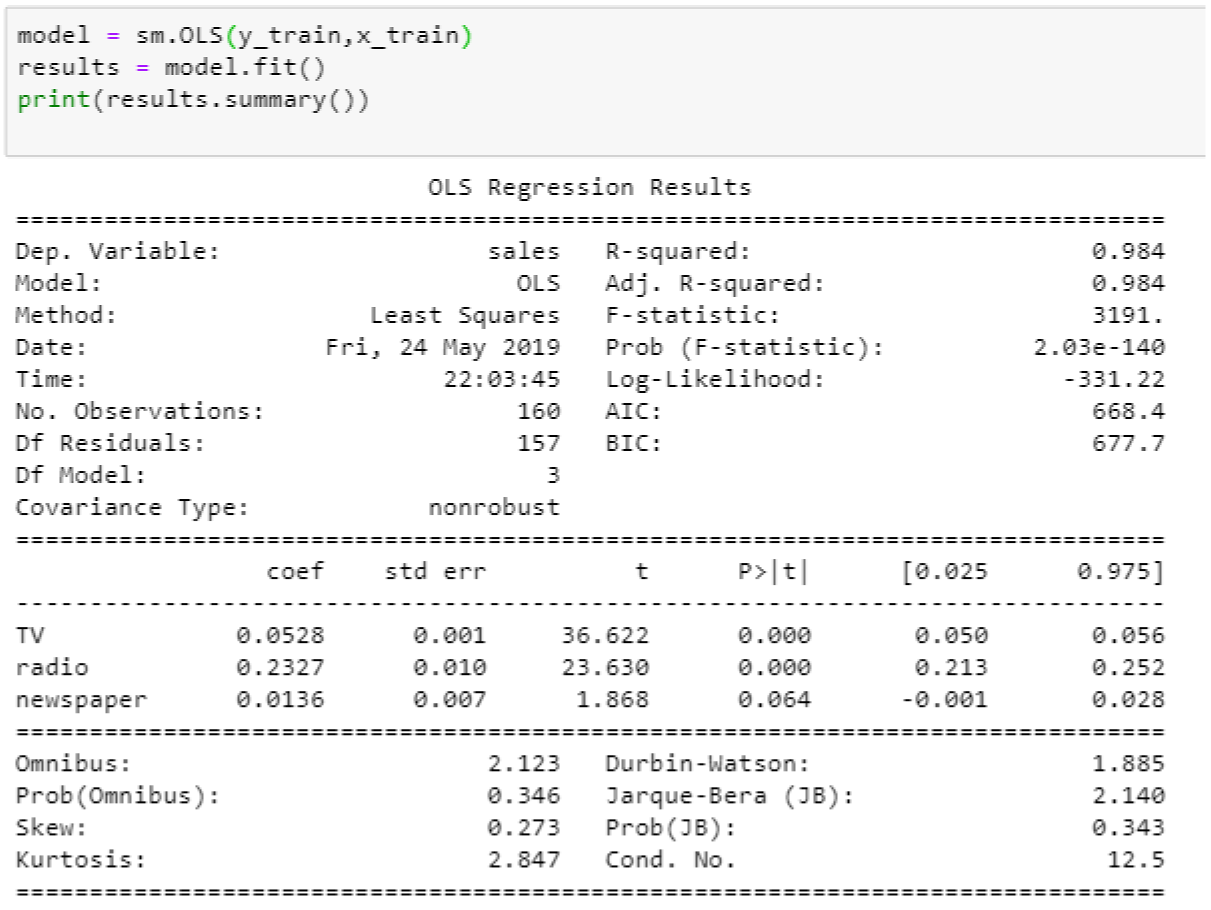
5. **Little or No autocorrelation in the residuals**

Autocorrelation occurs when the residual errors are dependent on each other. The presence of correlation in error terms drastically reduces model’s accuracy. This usually occurs in time series models where the next instant is dependent on previous instant.

Autocorrelation can be tested with the help of Durbin-Watson test.The null hypothesis of the test is that there is no serial correlation. The Durbin-Watson test statistics is defined as:

$$\sum_{t=2}^{T}((e_t - e_{t-1})^2) / \sum_{t=1}^{T} e_t^2$$

The test statistic is approximately equal to 2*(1-r) where r is the sample autocorrelation of the residuals. Thus, for r == 0, indicating no serial correlation, the test statistic equals 2. This statistic will always be between 0 and 4. The close to 0 the statistic, the more evidence for positive serial correlation. The closer to 4, the more evidence for negative serial correlation.



```
model = sm.OLS(y_train,x_train)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  sales   R-squared:                       0.984
Model:                            OLS   Adj. R-squared:                  0.984
Method:                 Least Squares   F-statistic:                     3191.
Date:                Fri, 24 May 2019   Prob (F-statistic):          2.03e-140
Time:                        22:03:45   Log-Likelihood:                -331.22
No. Observations:                 160   AIC:                             668.4
Df Residuals:                     157   BIC:                             677.7
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
TV             0.0528      0.001     36.622      0.000       0.050       0.056
radio          0.2327      0.010     23.630      0.000       0.213       0.252
newspaper      0.0136      0.007      1.868      0.064      -0.001       0.028
==============================================================================
Omnibus:                        2.123   Durbin-Watson:                   1.885
Prob(Omnibus):                  0.346   Jarque-Bera (JB):                2.140
Skew:                           0.273   Prob(JB):                        0.343
Kurtosis:                       2.847   Cond. No.                         12.5
==============================================================================
```

From the above summary note that the value of Durbin-Watson test is 1.885 quite close to 2 as said before when the value of Durbin-Watson is equal to 2, r takes the value 0 from the equation 2*(1-r),which in turn tells us that the residuals are not correlated.

What is the coefficient of correlation and the coefficient of determination?

In simple terms, Correlation Coefficient shows the extent of "linear" relationship between two variables. Whereas Coefficient of Determination is a measure of goodness of fit for the model under study, i.e. it shows how much variation is explained by the model with respect to the total variation present in the data set.

Co-efficient Correlation r:

- The quantity *r*, called the *linear correlation coefficient*, measures the strength and the direction of a linear relationship between two variables. The linear correlation coefficient is sometimes referred to as the *Pearson product moment correlation coefficient* in honor of its developer Karl Pearson.
- The mathematical formula for computing ***r*** is:

$$r = \frac{n\sum xy - \left(\sum x\right)\left(\sum y\right)}{\sqrt{n\left(\sum x^2\right) - \left(\sum x\right)^2}\sqrt{n\left(\sum y^2\right) - \left(\sum y\right)^2}}$$

where n is the number of pairs of data.

- The value of r is such that $-1 \leq r \leq +1$. The + and – signs are used for positive linear correlations and negative linear correlations, respectively.
- *Positive correlation:* If x and y have a strong positive linear correlation, r is close to +1. An r value of exactly +1 indicates a perfect positive fit. Positive values indicate a relationship between x and y variables such that as values for x increases, values for y also increase.
- *Negative correlation:* If x and y have a strong negative linear correlation, r is close to -1. An r value of exactly -1 indicates a perfect negative fit. Negative values indicate a relationship between x and y such that as values for x increase, values for y decrease.
- *No correlation:* If there is no linear correlation or a weak linear correlation, r is close to 0. A value near zero means that there is a random, nonlinear relationship between the two variables
- Note that r is a dimensionless quantity; that is, it does not depend on the units employed.
- A *perfect* correlation of ± 1 occurs only when the data points all lie exactly on a straight line. If $r = +1$, the slope of this line is positive. If $r = -1$, the slope of this line is negative.
- A correlation greater than 0.8 is generally described as *strong*, whereas a correlation less than 0.5 is generally described as *weak*. These values can vary based upon the "type" of data being examined. A study utilizing scientific data may require a stronger correlation than a study using social science data.

Coefficient of Determination, r^2 or R^2:

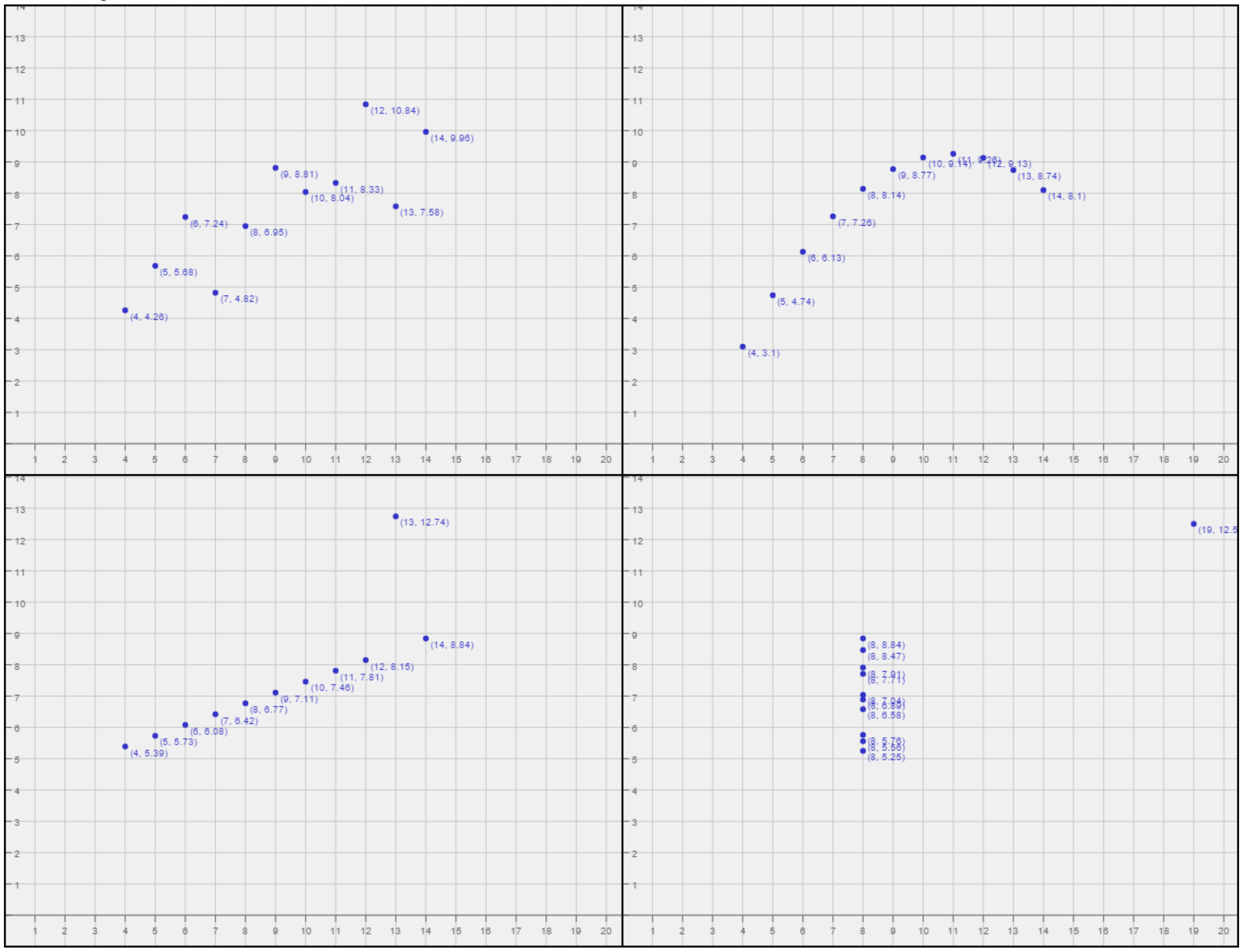
- The *coefficient of determination,* r^2, is useful because it gives the proportion of the variance (fluctuation) of one variable that is predictable from the other variable. It is a measure that allows us to determine how certain one can be in making predictions from a certain model/graph.
- The *coefficient of determination* is the ratio of the explained variation to the total variation.
- The *coefficient of determination* is such that $0 \leq r^2 \leq 1$, and denotes the strength of the linear association between x and y.
- The *coefficient of determination* represents the percent of the data that is the closest to the line of best fit. For example, if $r = 0.922$, then $r^2 = 0.850$, which means that 85% of the total variation in y can be explained by the linear relationship between x and y (as described by the regression equation). The other 15% of the total variation

in *y* remainsunexplained.

- The *coefficient of determination* is a measure of how well the regression line represents the data. If the regression line passes exactly through every point on the scatter plot, it would be able to explain all of the variation. The further the line is away from the points, the less it is able to explain.

Explain the Anscombe’s quartet in detail.

Statistics are great for describing general trends and aspects of data, but statistics alone can't fully depict any data set. Francis Anscombe realized this in 1973 and created several data sets, all with several identical statistical properties, to illustrate it. These data sets, collectively known as “Anscombe's Quartet,” are shown below.



All four of these data sets have the same variance in x, variance in y, mean of x, mean of y, and linear regression. But, as you can clearly tell, they are all quite different from one another. So, what does this mean to you as a statistician?

Well, to start, Anscombe's Quartet is a great demonstration of the importance of graphing data to analyze it. Given simply variance values, means, and even linear regressions can not accurately portray data in its native form. Anscombe's Quartet shows that multiple data sets with many similar statistical properties can still be vastly different from one another when graphed.

Additionally, Anscombe's Quartet warns of the dangers of outliers in data sets. Think about it: if the bottom two graphs didn't have that one point that strayed so far from all the other points, their statistical properties would no longer be identical to the two top graphs. In fact, their statistical properties would more accurately resemble the lines that the graphs seem to depict.
how to analyze your data. For example, while all four data sets have the same linear regression, it is obvious that the top right graph really shouldn't be analyzed with a linear regression at all because it's a curvature. Conversely, the top left graph probably *should* be analyzed with a linear regression because it's a scatter plot that moves in a roughly linear manner. These observations demonstrate the value in graphing your data before analyzing it.
Anscombe's Quartet reminds us that graphing data prior to analysis is good practice, outliers should be removed when analyzing data, and statistics about a data set do not fully depict the data set in its entirety.

```
>>> import pandas as pd  # Data manipulation
>>> import ciw  # The discrete event simulation library we will use to study queues
>>> import matplotlib.pyplot as plt  # Plots
>>> import seaborn as sns  # Powerful plots
>>> from scipy import stats  # Linear regression
>>> import numpy as np  # Quick summary statistics
>>> import tqdm  # A progress bar
```

Anscombe’s quartet

First of all, let us get the data set, that’s immediate to do as it comes ready loadable within the seaborn library:

```
>>> anscombe = sns.load_dataset("anscombe")
```

We can see that there are 4 data sets each with different values of xx and yy:

anscombe

```
  dataset    x     y
0       I  10.0  8.04
1       I   8.0  6.95
2       I  13.0  7.58
3       I   9.0  8.81
4       I  11.0  8.33
5       I  14.0  9.96
```

6	I	6.0	7.24
7	I	4.0	4.26
8	I	12.0	10.84
9	I	7.0	4.82
10	I	5.0	5.68
11	II	10.0	9.14
12	II	8.0	8.14
13	II	13.0	8.74
14	II	9.0	8.77
15	II	11.0	9.26
16	II	14.0	8.10
17	II	6.0	6.13
18	II	4.0	3.10
19	II	12.0	9.13
20	II	7.0	7.26
21	II	5.0	4.74
22	III	10.0	7.46
23	III	8.0	6.77
24	III	13.0	12.74
25	III	9.0	7.11
26	III	11.0	7.81
27	III	14.0	8.84
28	III	6.0	6.08

```
29     III   4.0   5.39

30     III  12.0   8.15

31     III   7.0   6.42

32     III   5.0   5.73

33      IV   8.0   6.58

34      IV   8.0   5.76

35      IV   8.0   7.71

36      IV   8.0   8.84

37      IV   8.0   8.47

38      IV   8.0   7.04

39      IV   8.0   5.25

40      IV  19.0  12.50

41      IV   8.0   5.56

42      IV   8.0   7.91

43      IV   8.0   6.89
```

Let us take a quick look at the summary measures:

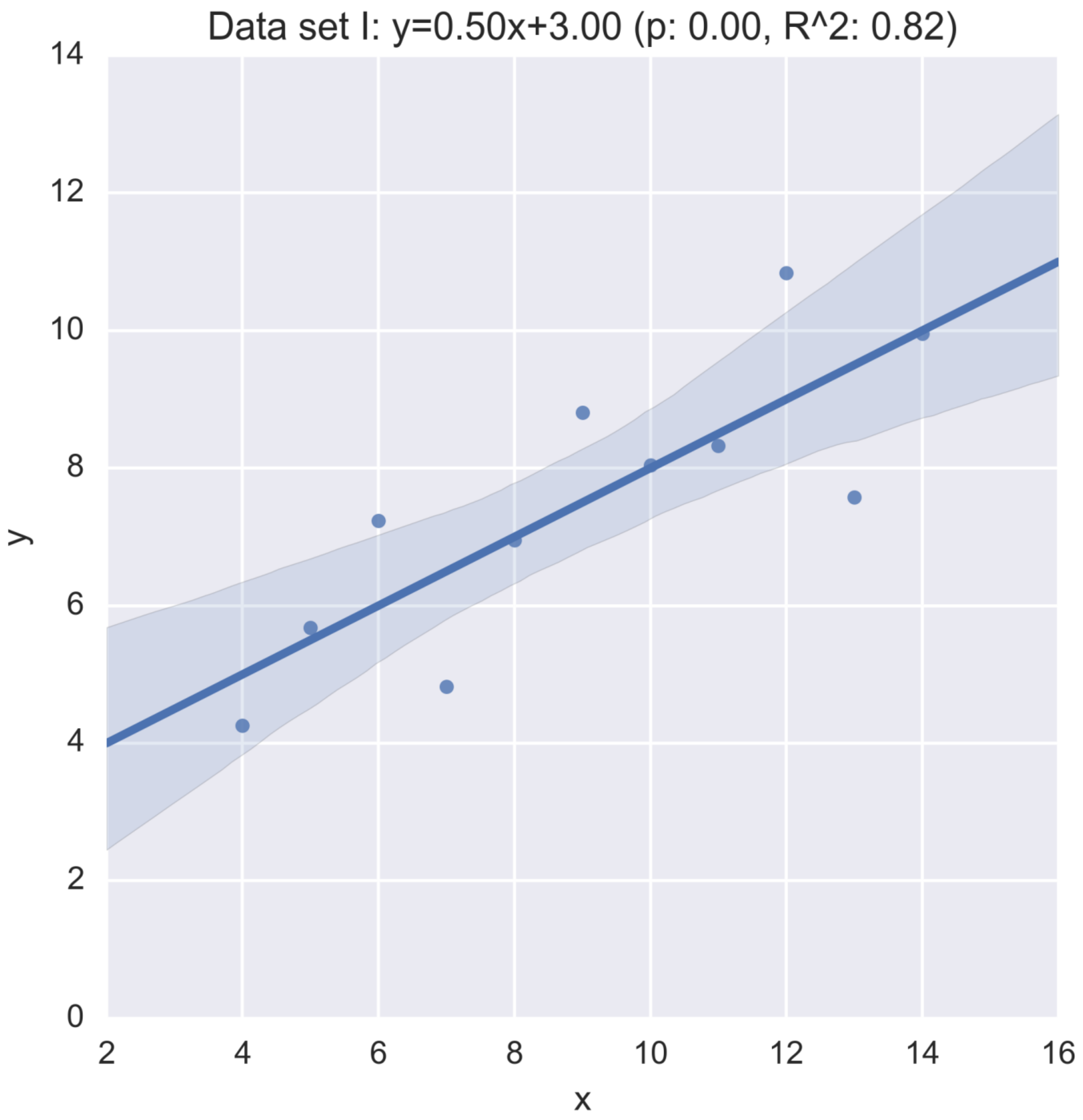
```
>>> anscombe.groupby("dataset").describe()

                      x          y

dataset

I       count  11.000000  11.000000

        mean    9.000000   7.500909

        std     3.316625   2.031568

        min     4.000000   4.260000
```

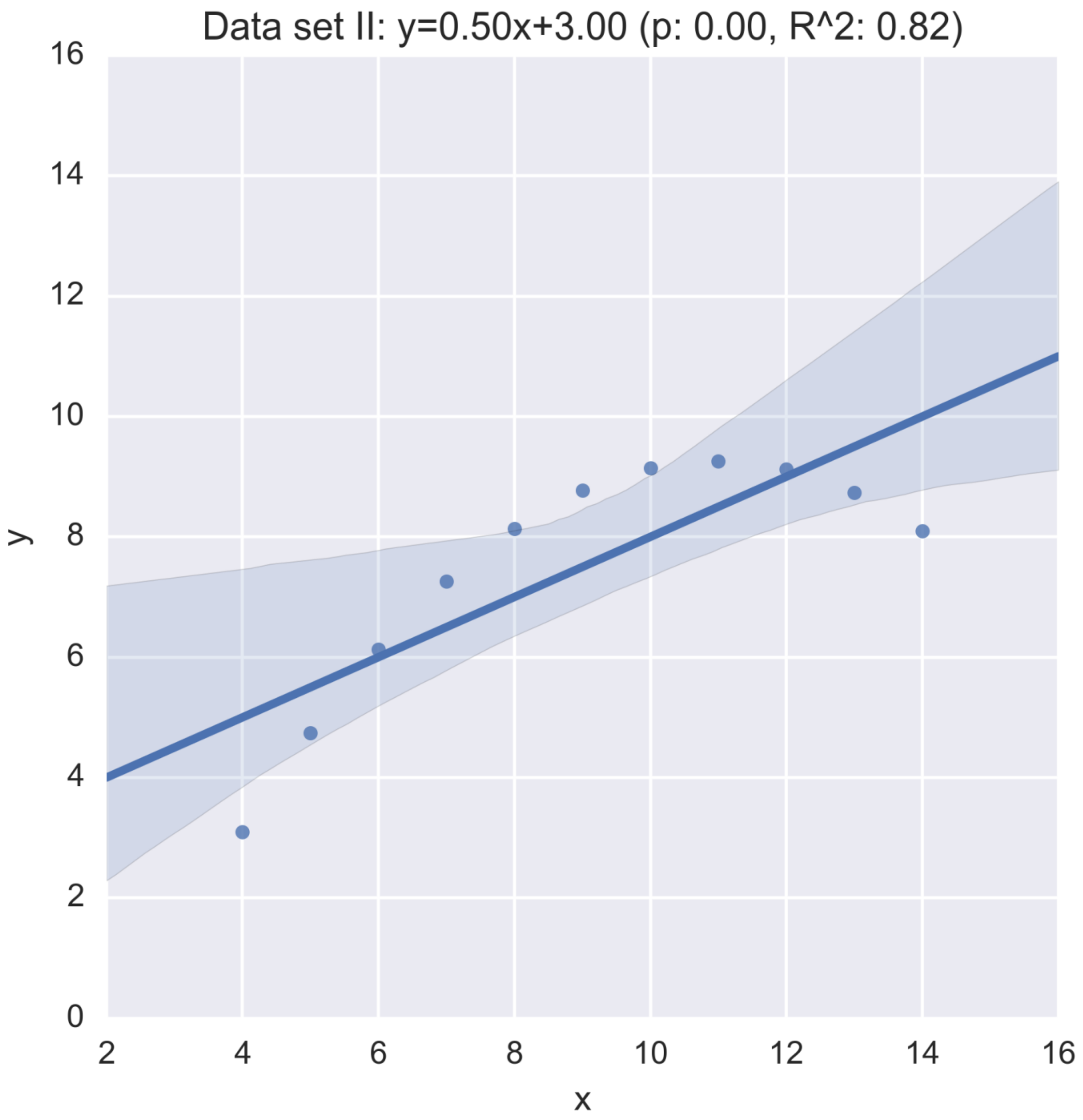
```
      25%     6.500000   6.315000
      50%     9.000000   7.580000
      75%    11.500000   8.570000
      max    14.000000  10.840000
II    count  11.000000  11.000000
      mean    9.000000   7.500909
      std     3.316625   2.031657
      min     4.000000   3.100000
      25%     6.500000   6.695000
      50%     9.000000   8.140000
      75%    11.500000   8.950000
      max    14.000000   9.260000
III   count  11.000000  11.000000
      mean    9.000000   7.500000
      std     3.316625   2.030424
      min     4.000000   5.390000
      25%     6.500000   6.250000
      50%     9.000000   7.110000
      75%    11.500000   7.980000
      max    14.000000  12.740000
IV    count  11.000000  11.000000
      mean    9.000000   7.500909
      std     3.316625   2.030579
```

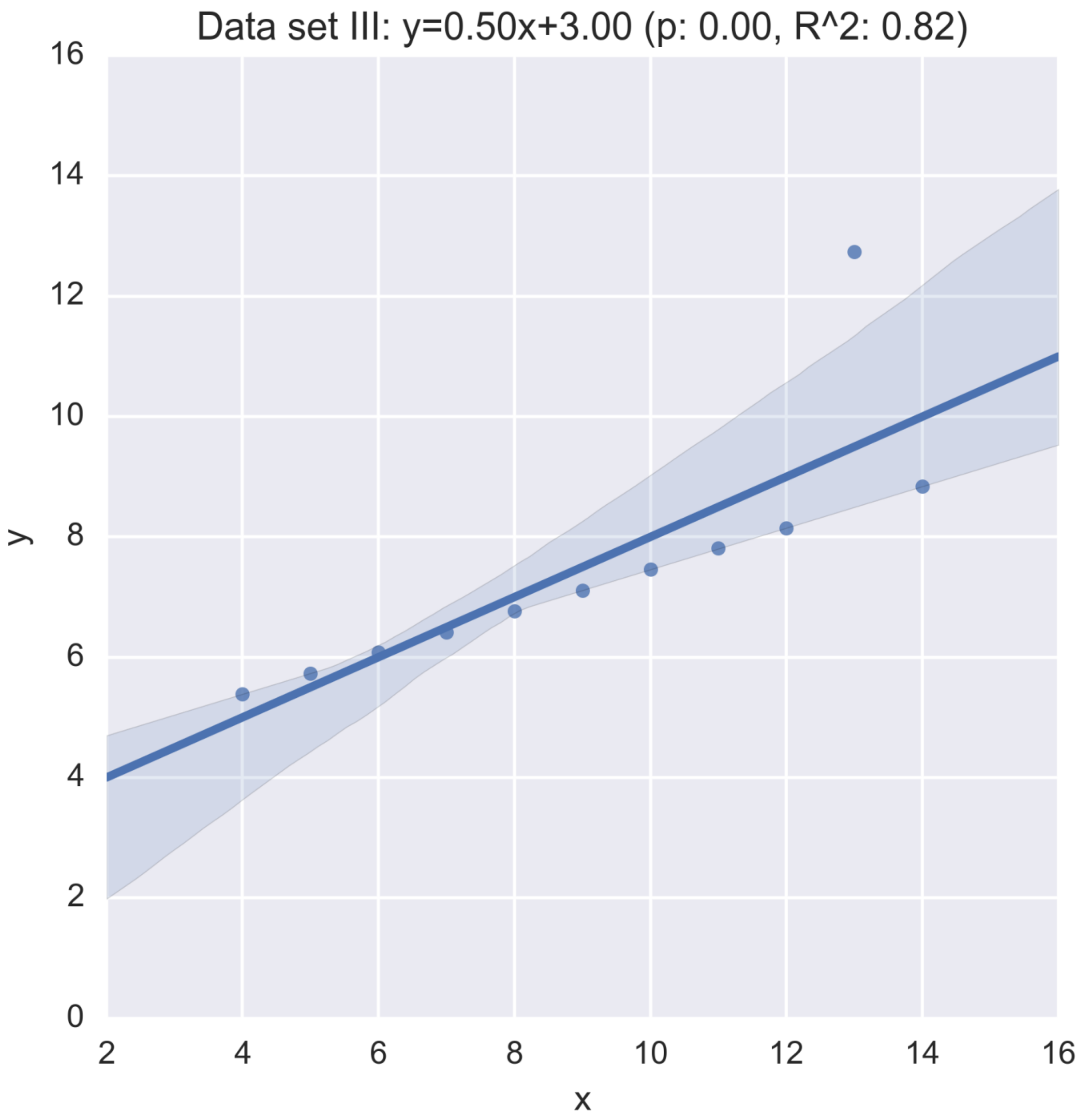
```
min     8.000000   5.250000

25%     8.000000   6.170000

50%     8.000000   7.040000

75%     8.000000   8.190000

max    19.000000  12.500000
```

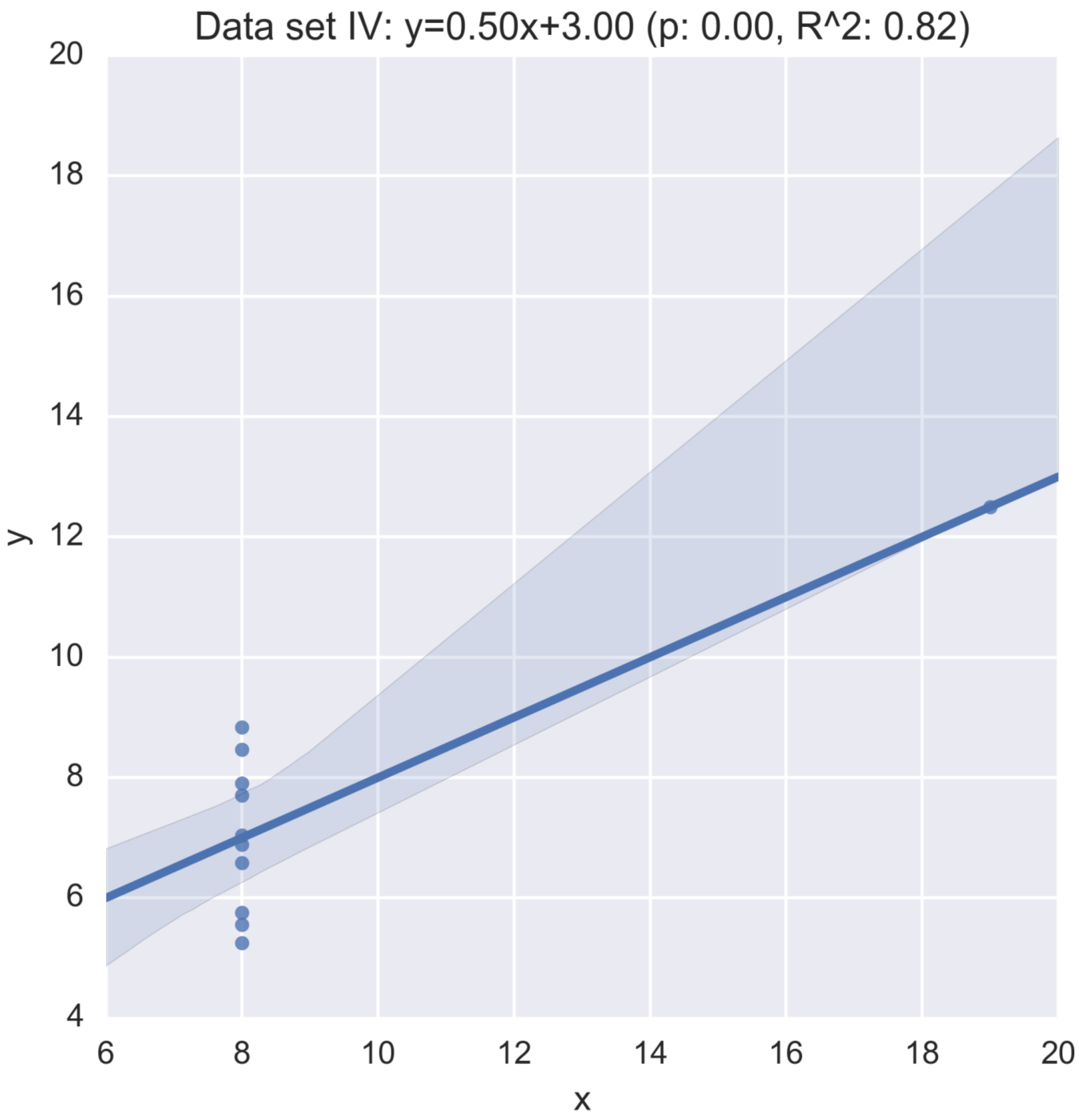
We see that the summary statistics are, if not the same (a mean in xx of 9, in yy of 7.5) **very similar**. We can also take a look at the relationship between xx and yy in each data set by computing the regression line and visualizing it:

```
for data_set in anscombe.dataset.unique():
...     df = anscombe.query("dataset == '{}'".format(data_set))
...     slope, intercept, r_val, p_val, slope_std_error = stats.linregress(x=df.x, y=df.y)
...     sns.lmplot(x="x", y="y", data=df);
...         plt.title("Data set {}: y={:.2f}x+{:.2f} (p: {:.2f}, R^2: {:.2f})".format(data_set, slope,
intercept, p_val, r_val))
```


Data set I: y=0.50x+3.00 (p: 0.00, R^2: 0.82)
y
x
0
2
4
6
8
10
12
14
16


Data set II: y=0.50x+3.00 (p: 0.00, R^2: 0.82)
y
x
0
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16


Data set III: y=0.50x+3.00 (p: 0.00, R^2: 0.82)
y
x
0
2
4
6
8
10
12
14
16
2
4
6
8
10
12
14
16



While the plots show that each data set is evidently different, they also all have the same regression line (with the same p,R2p,R2 values!):
y=0.50x+3y=0.50x+3
Anscombe’s quartet is often used as an example justifying that a summary of a data set will inherently lose information and so should be accompanied by further study/understanding (such as in this case: viewing the plots of the data).

What is Pearson’s R?

Correlation is a bi-variate analysis that measures the strength of association between two variables and the direction of the relationship. In terms of the strength of relationship, the value of the correlation coefficient varies between +1 and -1. A value of ± 1 indicates a perfect degree of association between the two variables. As the correlation coefficient value goes towards 0, the relationship between the two variables will be weaker. The direction of the relationship is indicated by the sign of the coefficient; a + sign indicates a positive relationship and a - sign indicates a negative relationship.
Usually, in statistics, we measure four types of correlations:

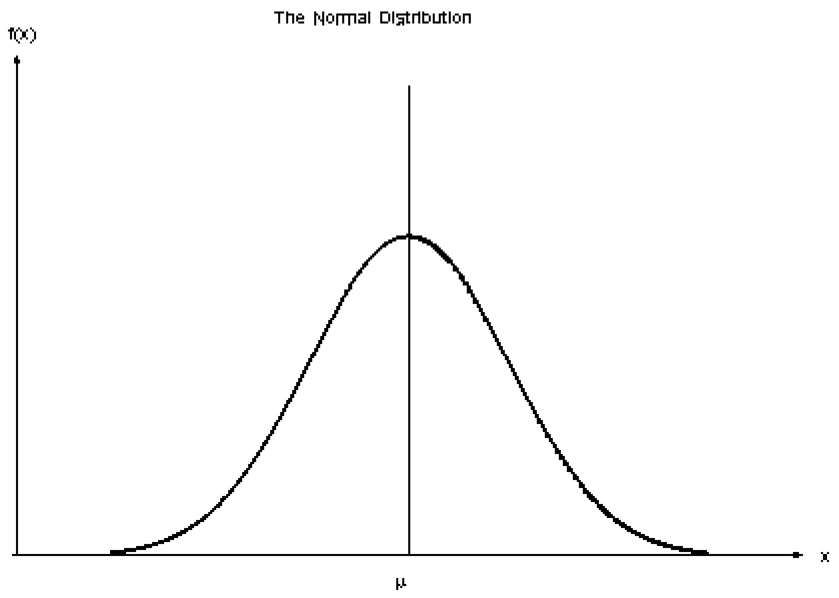
- Pearson correlation
- Kendall rank correlation
- Spearman correlation
- Point-Biserial correlation.

Pearson R Correlation
As the title suggests, we'll only cover Pearson correlation coefficient. I'll keep this short but very informative so you can go ahead and do this on your own. Pearson correlation coefficient is a measure of the strength of a linear association between two variables — denoted by r. You'll come across Pearson r correlation

Questions a Pearson correlation answers

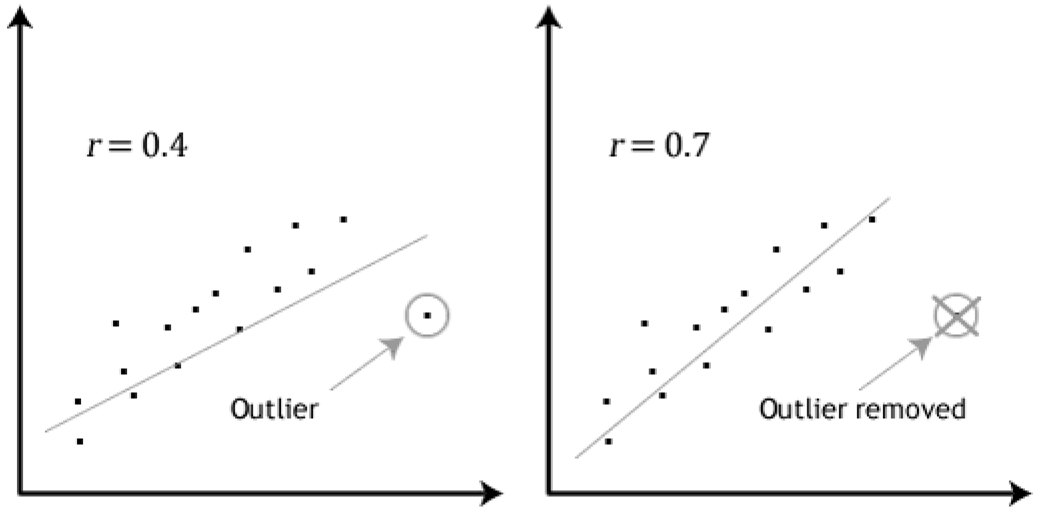
- Is there a statistically significant relationship between age and height?
- Is there a relationship between temperature and ice cream sales?
- Is there a relationship among job satisfaction, productivity, and income?
- Which two variable have the strongest co-relation between age, height, weight, size of family and family income?

Assumptions
1. For the Pearson r correlation, both variables should be **normally distributed**. i.e the normal distribution describes how the values of a variable are distributed. This is sometimes called the 'Bell Curve' or the 'Gaussian Curve'. A simple way to do this is to determine the normality of each variable separately using the Shapiro-Wilk Test.



Normal Distribution

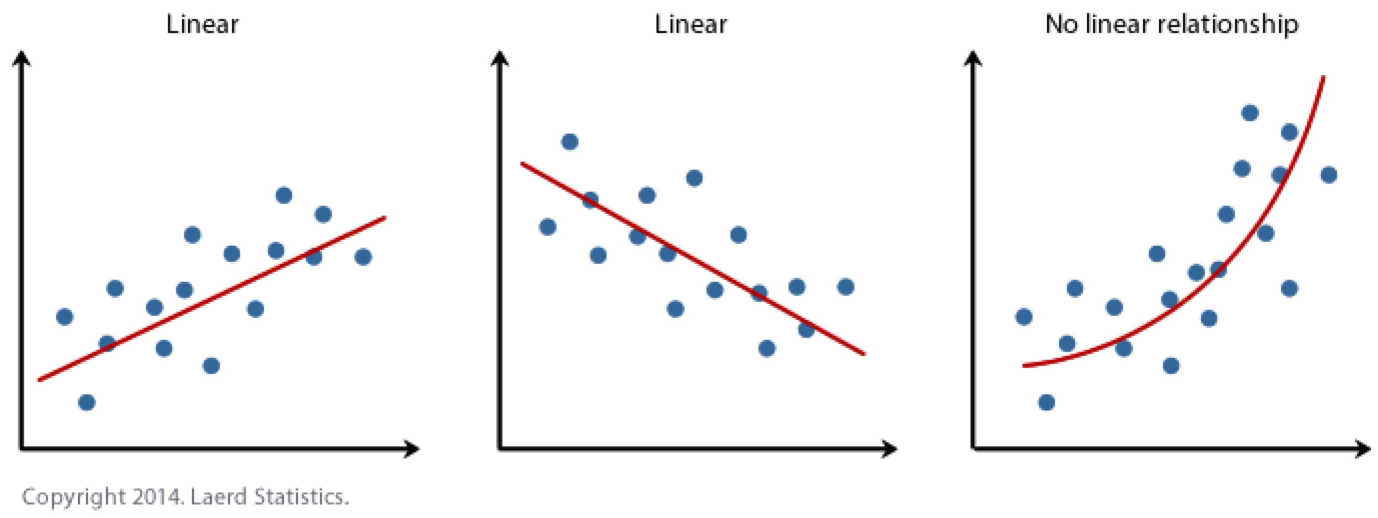
2. There should be **no significant outliers**. We all know what outliers are but we don't know the effect of outliers on Pearson's correlation coefficient, r. Pearson's correlation coefficient, r, is very sensitive to outliers, which can have a very large effect on the line of best fit and the Pearson correlation coefficient. This means — including outliers in your analysis can lead to misleading results.



Outliers

3. Each variable should be **continuous** i.e. interval or ratios for example weight, time, height, age etc. If one or both of the variables are ordinal in measurement, then a Spearman correlation could be conducted instead.
4. The two variables have a **linear relationship**. Scatter plots will help you tell whether the variables have a linear relationship. If the data points have a straight line (and not a curve), then

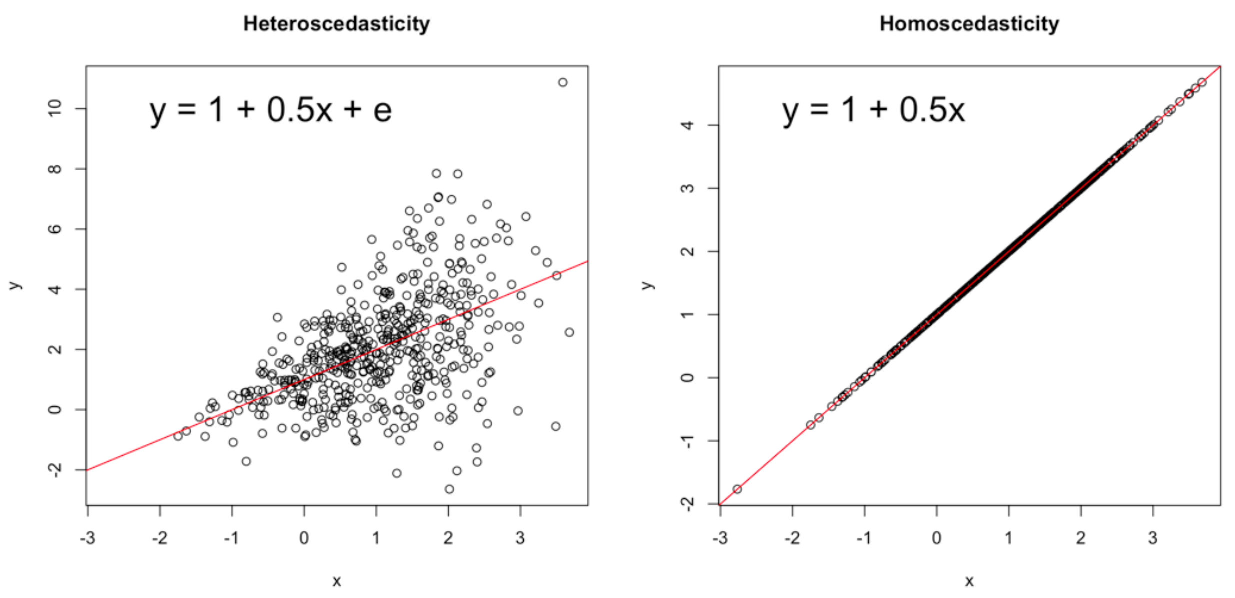
the data satisfies the linearity assumption. If the data you have is not linearly related you might have to run a non-parametric.



Linear and non-Linear Relationships

5. The observations are **paired observations.** That is, for every observation of the independent variable, there must be a corresponding observation of the dependent variable. For example if you're calculating the correlation between age and weight. If there are 12 observations of weight, you should have 12 observations of age. **i.e. no blanks.**

6. **Homoscedascity**. I've saved best for last. The hard is hard to pronounce but the concept is simple. Homoscedascity simply refers to '**equal variances**'. A scatter-plot makes it easy to check for this. If the points lie equally on both sides of the line of best fit, then the data is homoscedastic. As a bonus — the opposite of homoscedascity is heteroscedascity which refers to refers to the circumstance in which the variability of a variable is unequal across the range of values of a second variable that predicts it.



Heteroscedasticity v/s homoscedasticity

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
```

Let’s read the data and put it in a dataframe.

```
SuicideRate = pd.read_csv("suicide-rates-overview-1985-to-2016.csv")
```

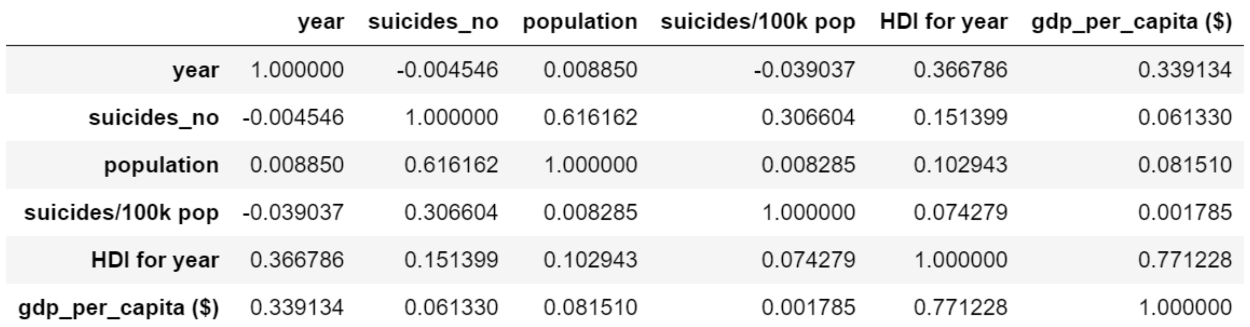
SuicideRate.head()

	country	year	sex	age	suicides_no	population	suicides/100k pop	country-year	HDI for year	gdp_for_year ($)	gdp_per_capita ($)	generation
0	Albania	1987	male	15-24 years	21	312900	6.71	Albania1987	NaN	2,156,624,900	796	Generation X
1	Albania	1987	male	35-54 years	16	308000	5.19	Albania1987	NaN	2,156,624,900	796	Silent
2	Albania	1987	female	15-24 years	14	289700	4.83	Albania1987	NaN	2,156,624,900	796	Generation X
3	Albania	1987	male	75+ years	1	21800	4.59	Albania1987	NaN	2,156,624,900	796	G.I. Generation
4	Albania	1987	male	25-34 years	9	274300	3.28	Albania1987	NaN	2,156,624,900	796	Boomers

Calculating the Pearson Corellation

We’ll use method = ‘pearson’ for the dataframe.corr since we want to calculate the pearson coefficient of correlation. Then we’ll print it out and see what’s what!

```
pearsoncorr = SuicideRate.corr(method='pearson')pearsoncorr
```

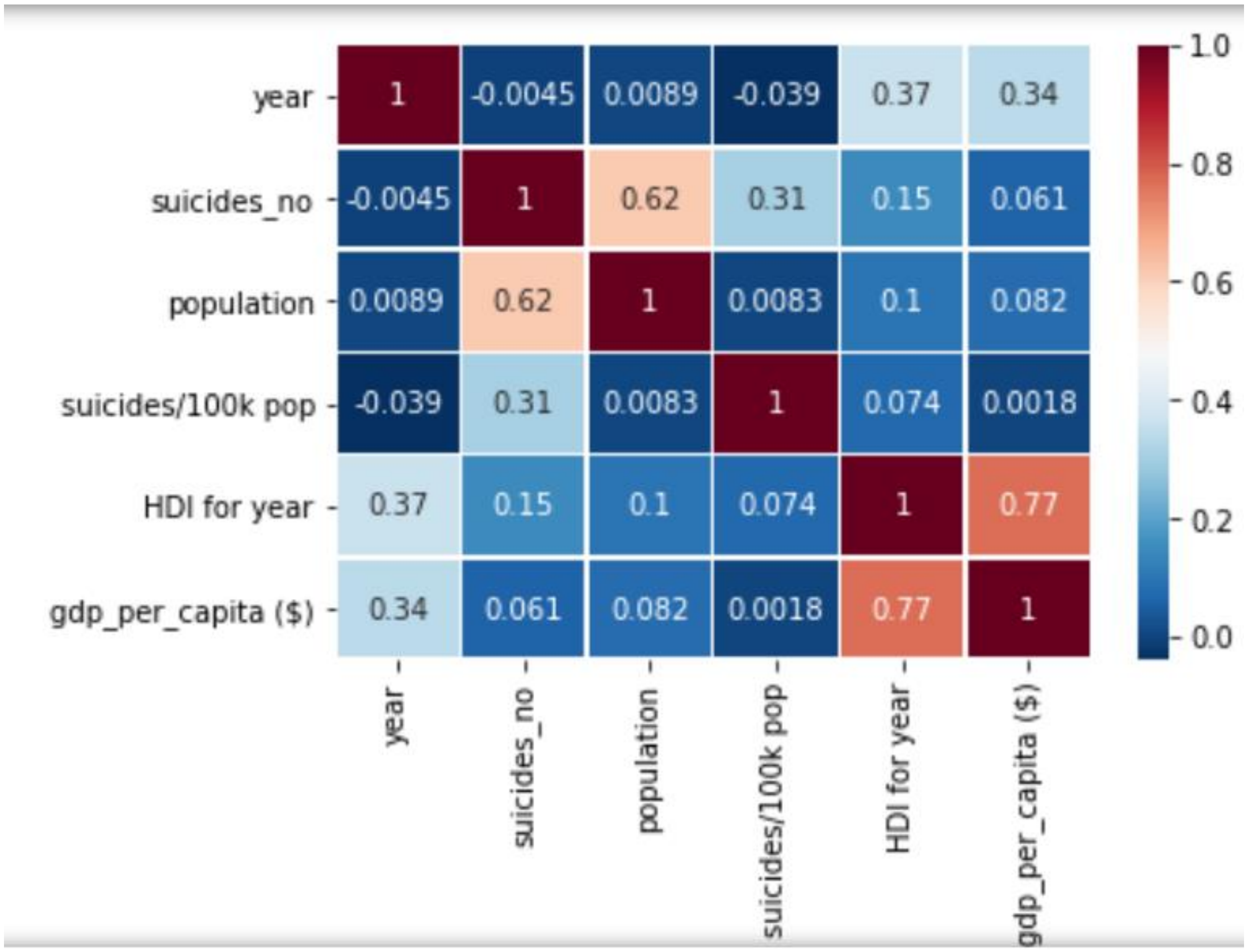


	year	suicides_no	population	suicides/100k pop	HDI for year	gdp_per_capita ($)
year	1.000000	-0.004546	0.008850	-0.039037	0.366786	0.339134
suicides_no	-0.004546	1.000000	0.616162	0.306604	0.151399	0.061330
population	0.008850	0.616162	1.000000	0.008285	0.102943	0.081510
suicides/100k pop	-0.039037	0.306604	0.008285	1.000000	0.074279	0.001785
HDI for year	0.366786	0.151399	0.102943	0.074279	1.000000	0.771228
gdp_per_capita ($)	0.339134	0.061330	0.081510	0.001785	0.771228	1.000000

To make this look beautiful and easier to interpret, add this after calculating the Pearson coefficient of correlation.

```
sb.heatmap(pearsoncorr, xticklabels=pearsoncorr.columns, yticklabels=pearsoncorr.columns,
cmap='RdBu_r',
annot=True,
linewidth=0.5)
```

Results and Interpreting the results.



A co-efficient close to 1 means that there’s a very strong positive correlation between the two variables. In our case, the maroon shows very strong correlations, the diagonal lines are the correlation of the variables to themselves — so they’ll obviously be 1.
Looking at this we can quickly see that”

- The Human development index (HDI) is strongly correlated to the GDP per Capita.
- The Population also has a strong correlation to the number of suicides. This is kind of what we’d expect right? A high population will have a higher number of suicides and vice versa.

What is scaling? Why is scaling performed? What is the difference between normalized scaling and standardized scaling?

Feature scaling (also known as data normalization) is the method used to standardize the range of features of data. Since, the range of values of data may vary widely, it becomes a necessary step in data preprocessing while using machine learning algorithms.

Scaling:

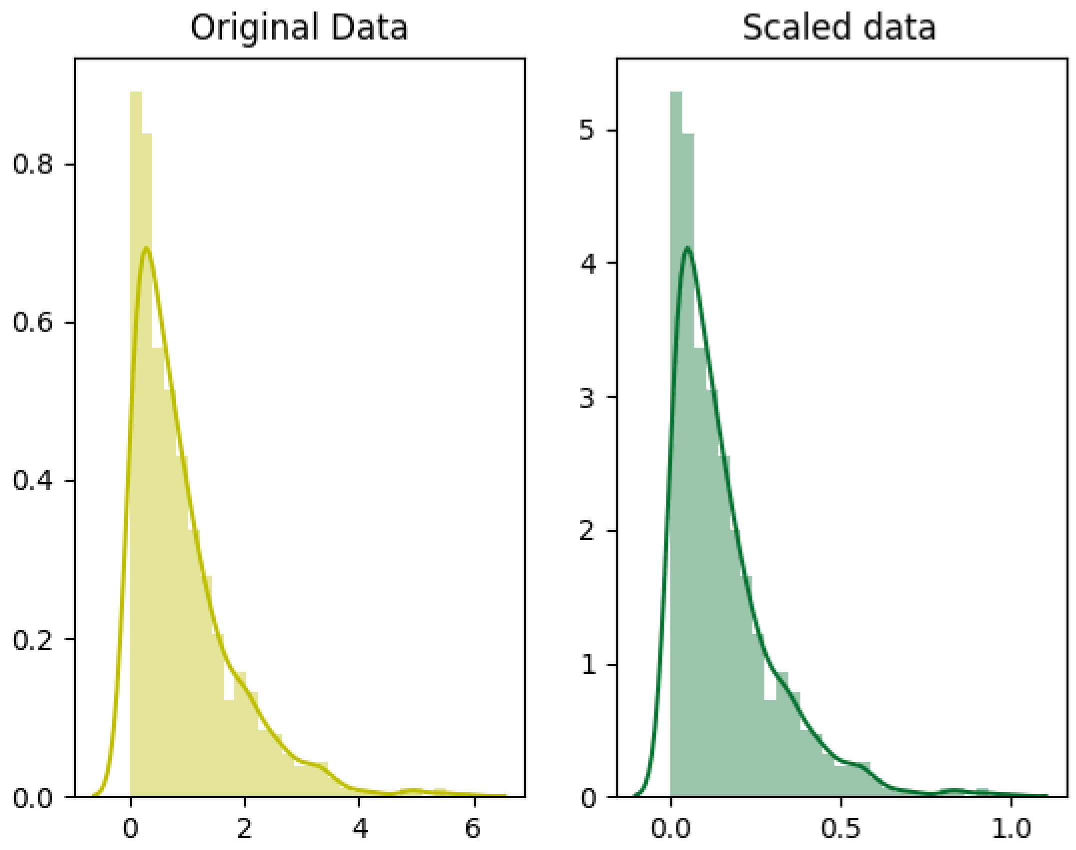
In scaling (also called min-max scaling), you transform the data such that the features are within a specific range e.g. [0, 1].

$x' = x - xmin/xmax - xmin$

where x’ is the normalized value.

Scaling is important in the algorithms such as support vector machines (SVM) and k-nearest neighbors (KNN) where distance between the data points is important. For example, in the dataset containing prices of products; without scaling, SVM might treat 1 USD equivalent to 1 INR though 1 USD = 65 INR.

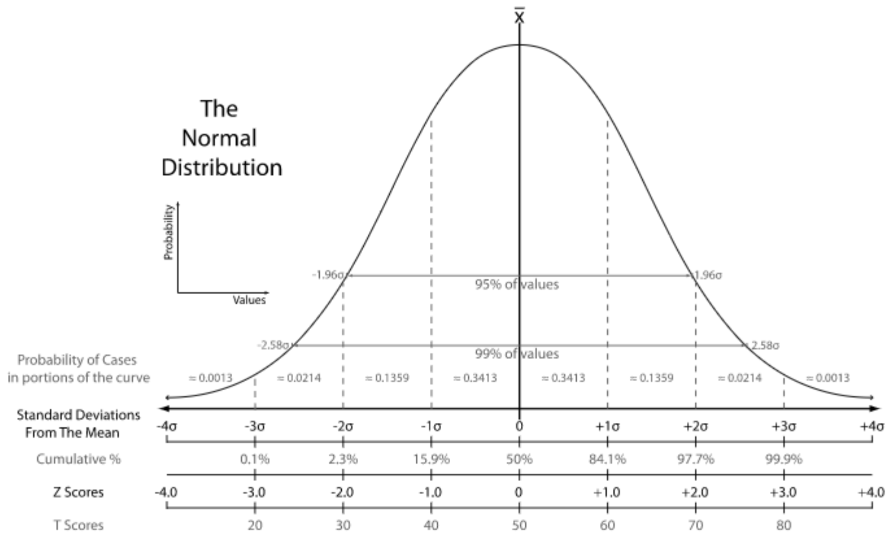
```
import matplotlib.pyplot as plt
from sklearn.preprocessing import minmax_scale

# set seed for reproducibility
np.random.seed(0)

# generate 1000 data points randomly drawn from an exponential distribution
original_data = np.random.exponential(size = 1000)

# mix-max scale the data between 0 and 1
scaled_data = minmax_scale(original_data)

# plot both together to compare
fig, ax=plt.subplots(1,2)
sns.distplot(original_data, ax=ax[0], color='y')
ax[0].set_title("Original Data")
sns.distplot(scaled_data, ax=ax[1])
ax[1].set_title("Scaled data")
plt.show()
```



Normalization and Standardization

The point of normalization is to change your observations so that they can be described as a normal distribution.

Normal distribution (Gaussian distribution), also known as the **bell curve**, is a specific statistical distribution where a roughly equal observations fall above and below the mean, the mean and the median are the same, and there are more observations closer to the mean.



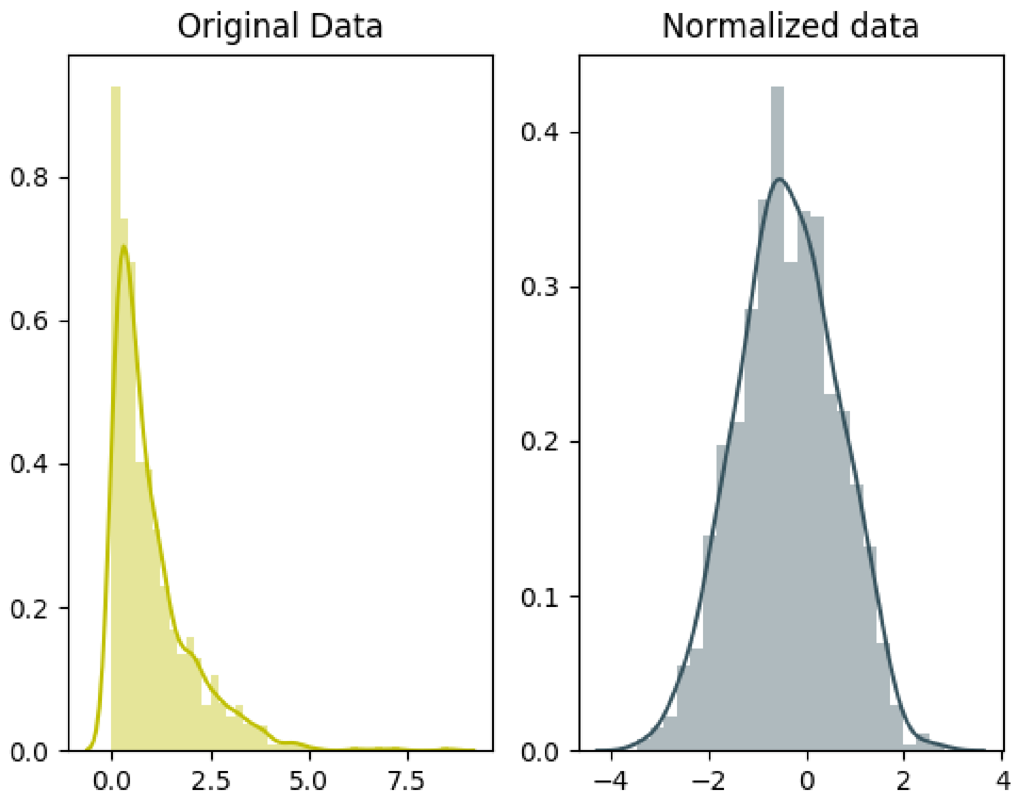
Note: *The above definition is as per statistics. There are various types of normalization. In fact, min-max scaling can also be said to a type of normalization. In machine learning, the following are most commonly used.*

1. **Standardization** *(also called* ***z-score normalization****)* transforms your data such that the resulting distribution has a mean of 0 and a standard deviation of 1. It's the definition that we read in the last paragraph.
   x′=x−xmean/σ

   where x is the original feature vector, xmeanxmean is the mean of that feature vector, and σ is its standard deviation.
   It's widely used in SVM, logistics regression and neural networks.

2. Simply called **normalization**, it's just another way of normalizing data. Note that, it's a different from min-max scaling in numerator, and from z-score normalization in the denominator.
   x′=x−xmeanxmax−xminx′=x−xmeanxmax−xmin
   For normalization, the maximum value you can get after applying the formula is 1, and the minimum value is 0. So all the values will be between 0 and 1.

```
# for Box-Cox Transformation
from scipy import stats

# normalize the exponential data with boxcox
normalized_data = stats.boxcox(original_data)

# plot both together to compare
fig, ax=plt.subplots(1,2)
sns.distplot(original_data, ax=ax[0], color='y')
ax[0].set_title("Original Data")
sns.distplot(normalized_data[0], ax=ax[1])
ax[1].set_title("Normalized data")
plt.show()
```



In scaling, you're changing the range of your data while in normalization you're mostly changing the shape of the distribution of your data.
You need to normalize our data if you're going use a machine learning or statistics technique that assumes that data is normally distributed e.g. t-tests, ANOVAs, linear regression, linear discriminant analysis (LDA) and Gaussian Naive Bayes.

Applications
In stochastic gradient descent, feature scaling can sometimes improve the convergence speed of the algorithm. In support vector machines, it can reduce the time to find support vectors.

You might have observed that sometimes the value of VIF is infinite. Why does this happen?

A variance inflation factor(VIF) detects multicollinearity in regression analysis. Multicollinearity is when there's correlation between predictors (i.e. independent variables) in a model; it's presence can adversely affect your regression results. The VIF estimates how much the variance of a regression coefficient is inflated due to multicollinearity in the model.

VIFs are usually calculated by software, as part of regression analysis. You'll see a VIF column as part of the output. VIFs are calculated by taking a predictor, and regressing it against every other predictor in the model. This gives you the R-squared values, which can then be plugged into the VIF formula. "i" is the predictor you're looking at (e.g. x1 or x2):
variance inflation factor

$$\mathrm{VIF} = \frac{1}{1 - R_i^2}$$

Variance inflation factors range from 1 upwards. The numerical value for VIF tells you (in decimal form) what percentage the variance (i.e. the standard error squared) is inflated for each coefficient. For example, a VIF of 1.9 tells you that the variance of a particular coefficient is 90% bigger than what you would expect if there was no multicollinearity — if there was no correlation with other predictors.

A **rule of thumb** for interpreting the variance inflation factor:

- 1 = not correlated.
- Between 1 and 5 = moderately correlated.
- Greater than 5 = highly correlated.

Exactly how large a VIF has to be before it causes issues is a subject of debate. What is known is that the more your VIF increases, the less reliable your regression results are going to be. In general, a VIF above 10 indicates high correlation and is cause for concern. Some authors suggest a more conservative level of 2.5 or above.
Sometimes a high VIF is no cause for concern at all. For example, you can get a high VIF by including products or powers from other variables in your regression, like x and x^2. If you have

high VIFs for dummy variables representing nominal variables with three or more categories, those are usually not a problem.
The VIF has a lower bound of 1 but no upper bound. Authorities differ on how high the VIF has to be to constitute a problem. Personally, I tend to get concerned when a VIF is greater than 2.50, which corresponds to an R^2 of .60 with the other variables.
Regardless of your criterion for what constitutes a high VIF, there are at least three situations in which a high VIF is not a problem and can be safely ignored:

1. **The variables with high VIFs are control variables, and the variables of interest do not have high VIFs.** Here's the thing about multicollinearity: it's only a problem for the variables that are collinear. It increases the standard errors of their coefficients, and it may make those coefficients unstable in several ways. But so long as the collinear variables are only used as control variables, and they are not collinear with your variables of interest, there's no problem. The coefficients of the variables of interest are not affected, and the performance of the control variables as controls is not impaired.
   Here's an example the sample consists of U.S. colleges, the dependent variable is graduation rate, and the variable of interest is an indicator (dummy) for public vs. private. Two control variables are average SAT scores and average ACT scores for entering freshmen. These two variables have a correlation above .9, which corresponds to VIFs of at least 5.26 for each of them. But the VIF for the public/private indicator is only 1.04. So there's no problem to be concerned about, and no need to delete one or the other of the two controls.
   2. The high VIFs are caused by the inclusion of powers or products of other variables. If you specify a regression model with both x and x^2, there's a good chance that those two variables will be highly correlated. Similarly, if your model has x, z, and xz, both x and z are likely to be highly correlated with their product. This is not something to be concerned about, however, because the p-value for xz is not affected by the multicollinearity. This is easily demonstrated: you can greatly reduce the correlations by "centering" the variables (i.e., subtracting their means) before creating the powers or the products. But the p-value for x^2 or for xz will be exactly the same, regardless of whether or not you center. And all the results for the other variables (including the R^2 but not including the lower-order terms) will be the same in either case. So the multicollinearity has no adverse consequences.
   3. The variables with high VIFs are indicator (dummy) variables that represent a categorical variable with three or more categories. If the proportion of cases in the reference category is small, the indicator variables will necessarily have high VIFs, even if the categorical variable is not associated with other variables in the regression model.
   Suppose, for example, that a marital status variable has three categories: currently married, never married, and formerly married. You choose formerly married as the reference category, with indicator variables for the other two. What happens is that the correlation between those two indicators gets more negative as the fraction of people in the reference category gets smaller. For example, if 45 percent of people are never

married, 45 percent are married, and 10 percent are formerly married, the VIFs for the married and never-married indicators will be at least 3.0.

Is this a problem? Well, it does mean that *p*-values for the indicator variables may be high. But the overall test that *all* indicators have coefficients of zero is unaffected by the high VIFs. And nothing else in the regression is affected. If you really want to avoid the high VIFs, just choose a reference category with a larger fraction of the cases. That may be desirable in order to avoid situations where none of the individual indicators is statistically significant even though the overall set of indicators is significant.

What is the Gauss-Markov theorem?

A theorem that proves that if the error terms in a multiple regression have the same variance and are uncorrelated, then the estimators of the parameters in the model produced by least squares estimation are better (in the sense of having lower dispersion about the mean) than any other unbiased linear estimator.

Gauss Markov Assumptions

There are five Gauss Markov assumptions (also called *conditions*):

1. **Linearity**: the parameters we are estimating using the OLS method must be themselves linear.
2. **Random**: our data must have been randomly sampled from the population.
3. **Non-Collinearity**: the regressors being calculated aren't perfectly correlated with each other.
4. **Exogeneity**: the regressors aren't correlated with the error term.
5. **Homoscedasticity**: no matter what the values of our regressors might be, the error of the variance is constant.

   Purpose of the Assumptions

The **Gauss Markov assumptions** guarantee the validity of ordinary least squares for estimating regression coefficients.

Checking how well our data matches these assumptions is an important part of estimating regression coefficients. When you know where these conditions are violated, you may be able to plan ways to change your experiment setup to help your situation fit the ideal Gauss Markov situation more closely.

In practice, the Gauss Markov assumptions are **rarely all met perfectly**, but they are still useful as a benchmark, and because they show us what 'ideal' conditions would be. They also allow us to pinpoint problem areas that might cause our estimated regression coefficients to be inaccurate or even unusable.

Explain the gradient descent algorithm in detail.

Optimization refers to the task of minimizing/maximizing an objective function f(x) parameterized by x. In machine/deep learning terminology, it's the task of minimizing the cost/loss function J(w) parameterized by the model's parameters w ∈ R^d. Optimization algorithms (in case of minimization) have one of the following goals:

Find the global minimum of the objective function. This is feasible if the objective function is convex, i.e. any local minimum is a global minimum.
Find the lowest possible value of the objective function within its neighborhood. That's usually the case if the objective function is not convex as the case in most deep learning problems.
There are three kinds of optimization algorithms:

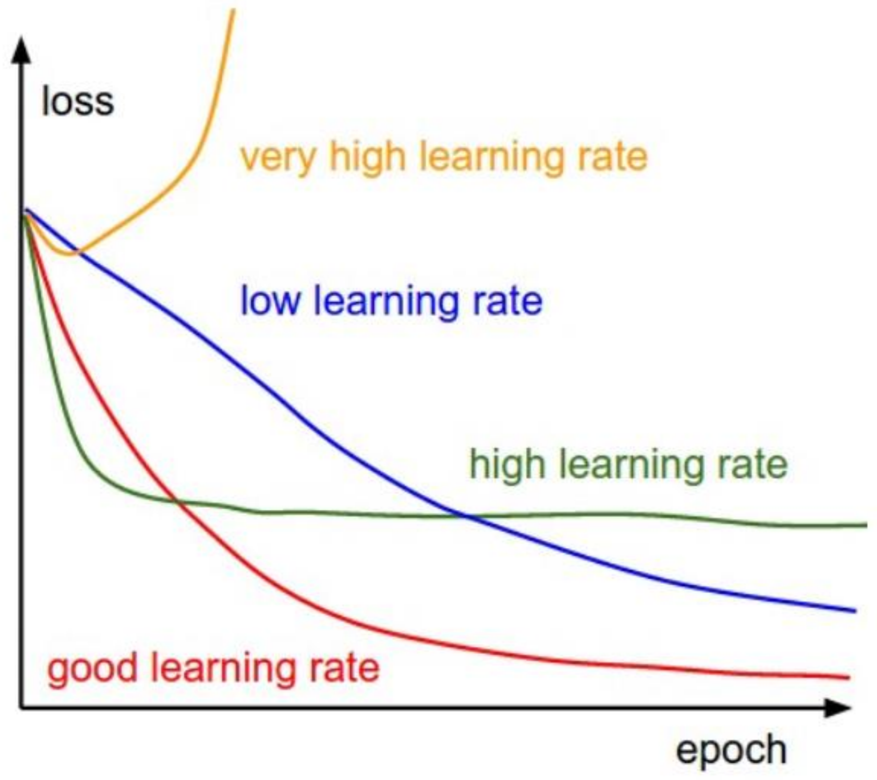
- Optimization algorithm that is not iterative and simply solves for one point.
- Optimization algorithm that is iterative in nature and converges to acceptable solution regardless of the parameters initialization such as gradient descent applied to logistic regression.
- Optimization algorithm that is iterative in nature and applied to a set of problems that have non-convex cost functions such as neural networks. Therefore, parameters' initialization plays a critical role in speeding up convergence and achieving lower error rates.

Gradient Descent is the most common optimization algorithm in *machine learning* and *deep learning*. It is a first-order optimization algorithm. This means it only takes into account the first derivative when performing the updates on the parameters. On each iteration, we update the parameters in the opposite direction of the gradient of the objective function *J(w)* w.r.t the parameters where the gradient gives the direction of the steepest ascent. The size of the step we take on each iteration to reach the local minimum is determined by the learning rate α. Therefore, we follow the direction of the slope downhill until we reach a local minimum.

let's assume that the logistic regression model has only two parameters: weight *w* and bias *b*.
1. Initialize weight *w* and bias *b* to any random numbers.
2. Pick a value for the learning rate α. The learning rate determines how big the step would be on each iteration.

- If α is very small, it would take long time to converge and become computationally expensive.
- If α is large, it may fail to converge and overshoot the minimum.

Therefore, plot the cost function against different values of α and pick the value of α that is right before the first value that didn't converge so that we would have a very fast learning algorithm that converges (see figure 2).



- The most commonly used rates are : *0.001, 0.003, 0.01, 0.03, 0.1, 0.3*.

3. Make sure to scale the data if it's on a very different scales. If we don't scale the data, the level curves (contours) would be narrower and taller which means it would take longer time to converge (see figure 3).

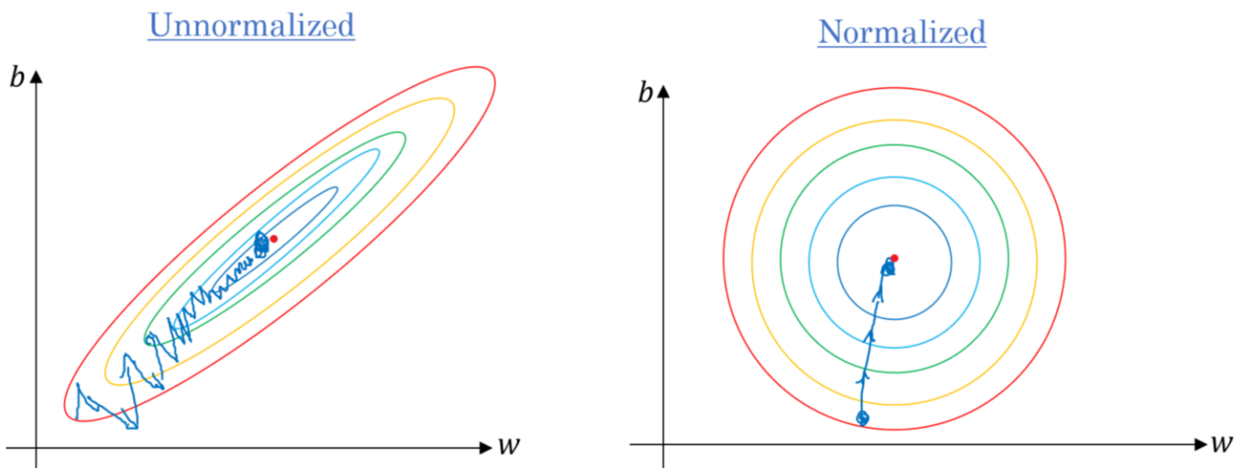


Figure 3: Gradient descent: normalized versus unnormalized level curves.

Scale the data to have μ = 0 and σ = 1. Below is the formula for scaling each example:

$$\frac{x_i - \mu}{\sigma} \tag{1}$$

4. On each iteration, take the partial derivative of the cost function *J(w)* w.r.t each parameter (gradient):

$$\frac{\partial}{\partial w} J(w) = \nabla_w J \tag{2}$$

$$\frac{\partial}{\partial b} J(w) = \nabla_b J \tag{3}$$

The update equations are:

$$w = w - \alpha \nabla_w J \tag{4}$$

$$b = b - \alpha \nabla_b J \tag{5}$$

For the sake of illustration, let's assume we don't have bias. If the slope of the current value of *w* > *0*, this means that we are to the right of optimal *w**. Therefore, the update will be negative, and will start getting close to the optimal values of *w**. However, if it's negative, the update will be positive and will increase the current values of *w* to converge to the optimal values of *w**(see figure 4):

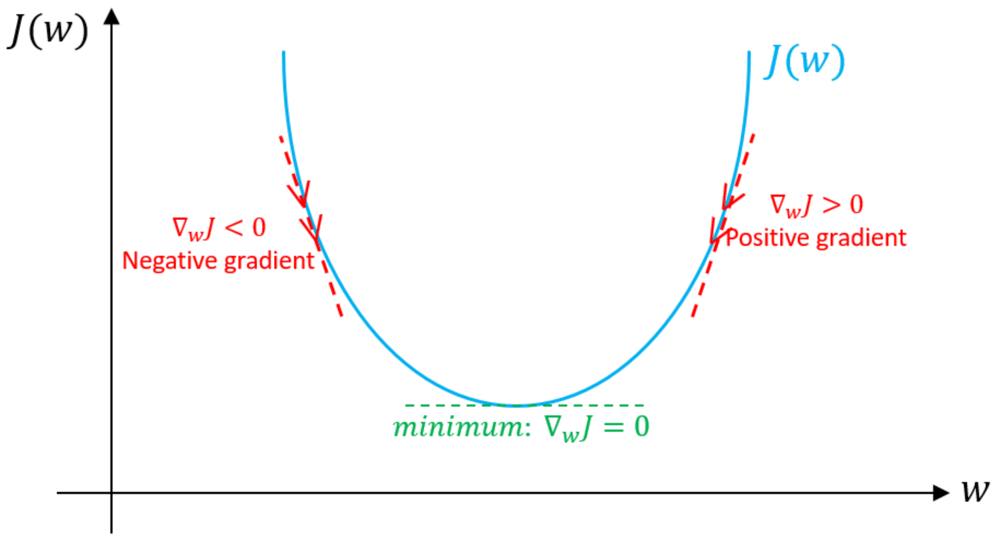


Figure 4: Gradient descent.

An illustration of how gradient descent algorithm uses the first derivative of the loss function to follow downhill it's minimum.

- Continue the process until the cost function converges. That is, until the error curve becomes flat and doesn't change.
- In addition, on each iteration, the step would be in the direction that gives the maximum change since it's perpendicular to level curves at each step.

Now let's discuss the three variants of gradient descent algorithm. The main difference between them is the amount of data we use when computing the gradients for each learning step. The trade-off between them is the accuracy of the gradient versus the time complexity to perform each parameter's update (learning step).

Batch Gradient Descent

Batch Gradient Descent is when we sum up over all examples on each iteration when performing the updates to the parameters. Therefore, for each update, we have to sum over all examples:

$$w = w - \alpha \nabla_w J(w) \tag{6}$$

```
for i in range(num_epochs):
    grad = compute_gradient(data, params)
    params = params — learning_rate * grad
```

The main advantages:

- We can use fixed learning rate during training without worrying about learning rate decay.
- It has straight trajectory towards the minimum and it is guaranteed to converge in theory to the global minimum if the loss function is convex and to a local minimum if the loss function is not convex.
- It has unbiased estimate of gradients. The more the examples, the lower the standard error.

The main disadvantages:

- Even though we can use vectorized implementation, it may still be slow to go over all examples especially when we have large datasets.
- Each step of learning happens after going over all examples where some examples may be redundant and don't contribute much to the update.

Mini-batch Gradient Descent

Instead of going over all examples, Mini-batch Gradient Descent sums up over lower number of examples based on the batch size. Therefore, learning happens on each mini-batch of *b* examples:

$$w = w - \alpha \nabla_w J(x^{\{i:i+b\}}, y^{\{i:i+b\}}; w) \tag{7}$$

- Shuffle the training data set to avoid pre-existing order of examples.
- Partition the training data set into *b* mini-batches based on the batch size. If the training set size is not divisible by batch size, the remaining will be its own batch.

```
for i in range(num_epochs):
    np.random.shuffle(data)
    for batch in radom_minibatches(data, batch_size=32):
        grad = compute_gradient(batch, params)
        params = params — learning_rate * grad
```

The batch size is something we can tune. It is usually chosen as power of 2 such as 32, 64, 128, 256, 512, etc. The reason behind it is because some hardware such as GPUs achieve better run time with common batch sizes such as power of 2.

The main advantages:

- Faster than Batch version because it goes through a lot less examples than Batch (all examples).
- Randomly selecting examples will help avoid redundant examples or examples that are very similar that don't contribute much to the learning.
- With batch size < size of training set, it adds noise to the learning process that helps improving generalization error.
- Even though with more examples the estimate would have lower standard error, the return is less than linear compared to the computational burden we incur.

The main disadvantages:

- It won't converge. On each iteration, the learning step may go back and forth due to the noise. Therefore, it wanders around the minimum region but never converges.
- Due to the noise, the learning steps have more oscillations (see figure 4) and requires adding learning-decay to decrease the learning rate as we become closer to the minimum.

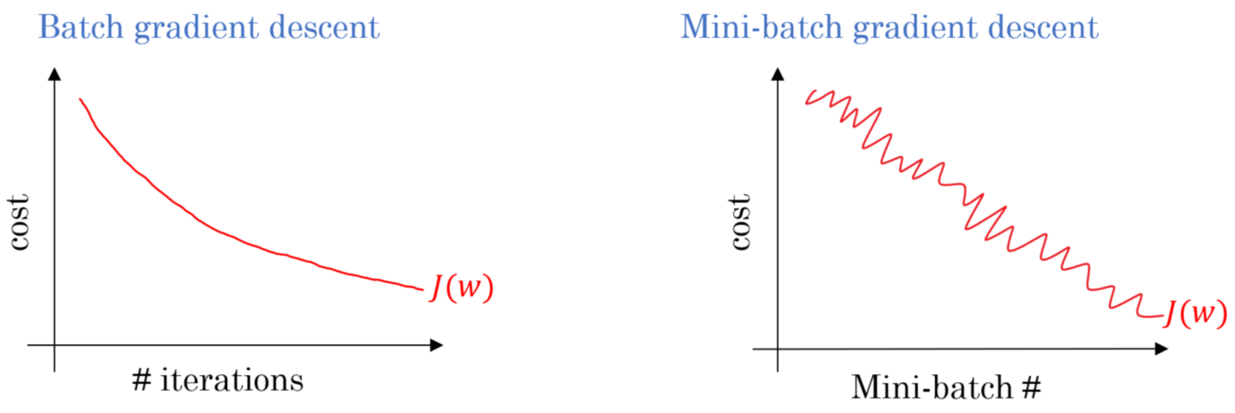


Figure 5: Gradient descent: batch versus mini-batch loss function

With large training datasets, we don't usually need more than 2–10 passes over all training examples (epochs). Note: with batch size $b = m$ (number of training examples), we get the Batch Gradient Descent.

Stochastic Gradient Descent

Instead of going through all examples, Stochastic Gradient Descent (SGD) performs the parameters update on each example (x^i,y^i). Therefore, learning happens on every example:

$$w = w - \alpha \nabla_w J(x^i, y^i; w) \tag{7}$$

- Shuffle the training data set to avoid pre-existing order of examples.
- Partition the training data set into *m* examples.

```
for i in range(num_epochs):
    np.random.shuffle(data)
    for example in data:
        grad = compute_gradient(example, params)
        params = params — learning_rate * grad
```

It shares most of the advantages and the disadvantages with mini-batch version. Below are the ones that are specific to SGD:

- It adds even more noise to the learning process than mini-batch that helps improving generalization error. However, this would increase the run time.

- We can’t utilize vectorization over 1 example and becomes very slow. Also, the variance becomes large since we only use 1 example for each learning step.

Below is a graph that shows the gradient descent’s variants and their direction towards the minimum:

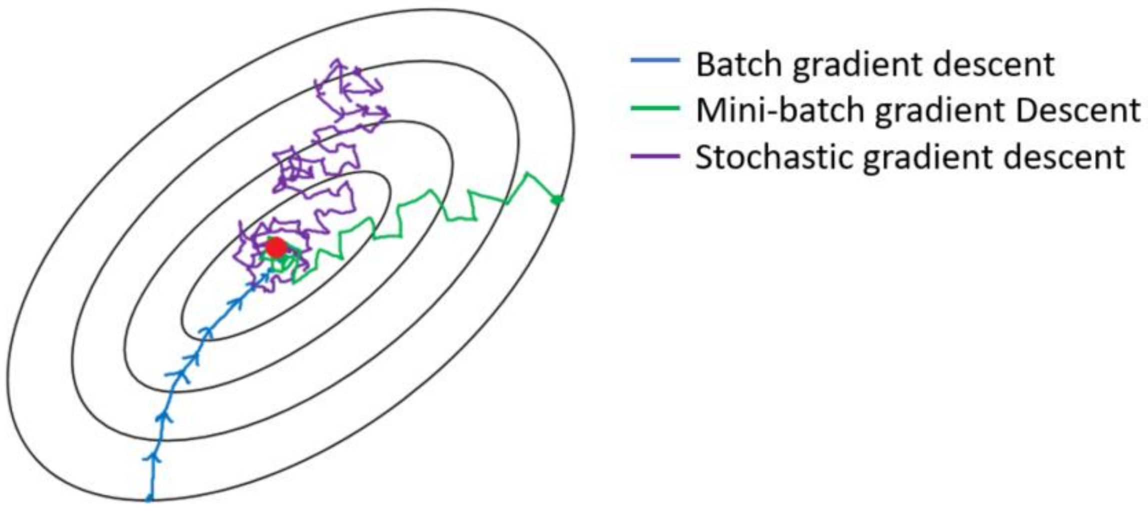


Figure 6: Gradient descent variants’ trajectory towards minimum

As the figure above shows, SGD direction is very noisy compared to mini-batch.

Challenges

Below are some challenges regarding gradient descent algorithm in general as well as its variants — mainly batch and mini-batch:

- Gradient descent is a first-order optimization algorithm, which means it doesn't take into account the second derivatives of the cost function. However, the curvature of the function affects the size of each learning step. The gradient measures the steepness of the curve but the second derivative measures the curvature of the curve. Therefore, if:

1. Second derivative = 0 →the curvature is linear. Therefore, the step size = the learning rate α.
2. Second derivative > 0 → the curvature is going upward. Therefore, the step size < the learning rate α and may lead to divergence.
3. Second derivative < 0 → the curvature is going downward. Therefore, the step size > the learning rate α.

As a result, the direction that looks promising to the gradient may not be so and may lead to slow the learning process or even diverge.

- If Hessian matrix has poor conditioning number, i.e. the direction of the most curvature has much more curvature than the direction of the lowest curvature. This will lead the cost function to be very sensitive in some directions and insensitive in other directions. As a result, it will make it harder on the gradient because the direction that looks promising for the gradient may not lead to big changes in the cost function (see figure 7).

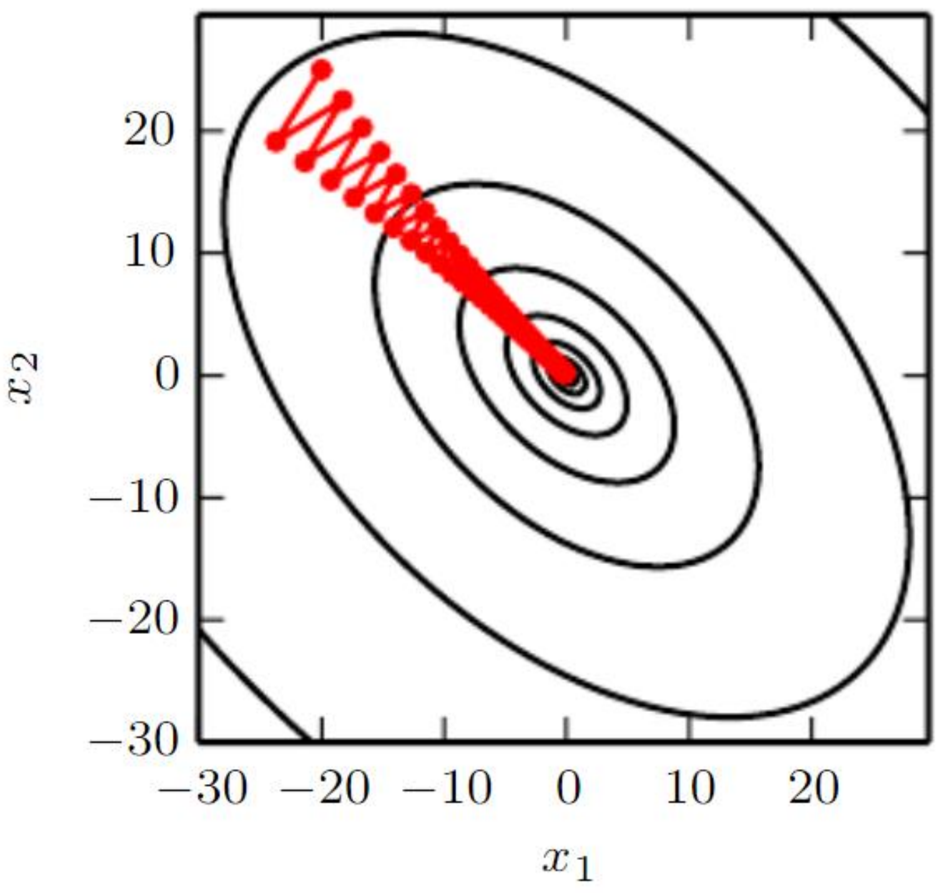


Figure 7: Gradient descent fails to exploit the curvature information contained in the Hessian matrix. Source

- The norm of the gradient gTg is supposed to decrease slowly with each learning step because the curve is getting flatter and steepness of the curve will decrease. However, we see that the norm of the gradient is increasing, because of the curvature of the curve. Nonetheless, even though the gradients' norm is increasing, we're able to achieve a very low error rates (see figure 8).

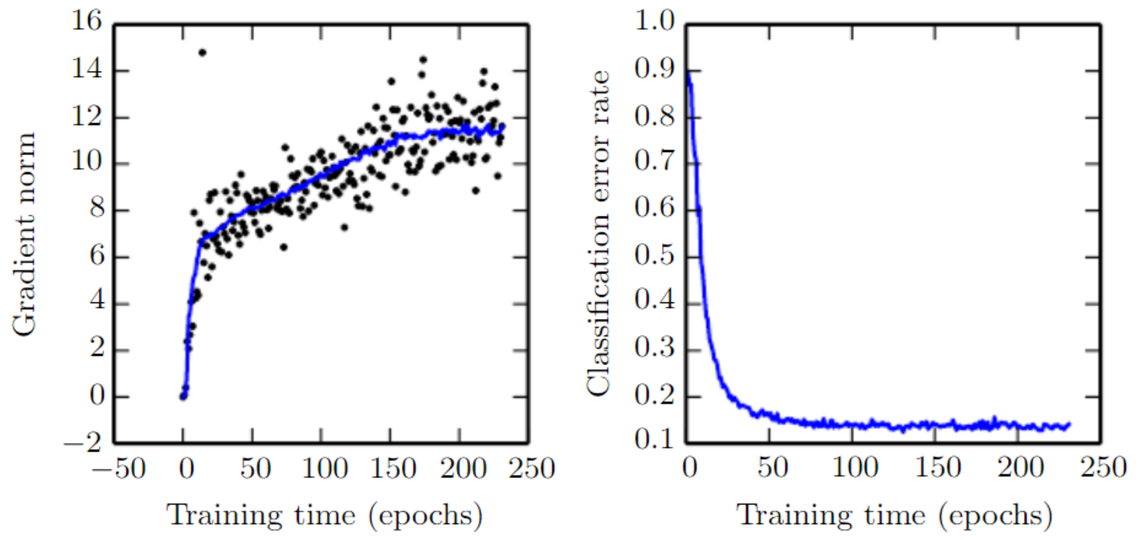


Figure 8: Gradient norm. Source

- In small dimensions, local minimum is common; however, in large dimensions, saddle points are more common. Saddle point is when the function curves up in some directions and curves down in other directions. In other words, saddle point looks like a minimum from one direction and a maximum from other direction (see figure 9). This happens when at least one eigenvalue of the hessian matrix is negative and the rest of eigenvalues are positive.

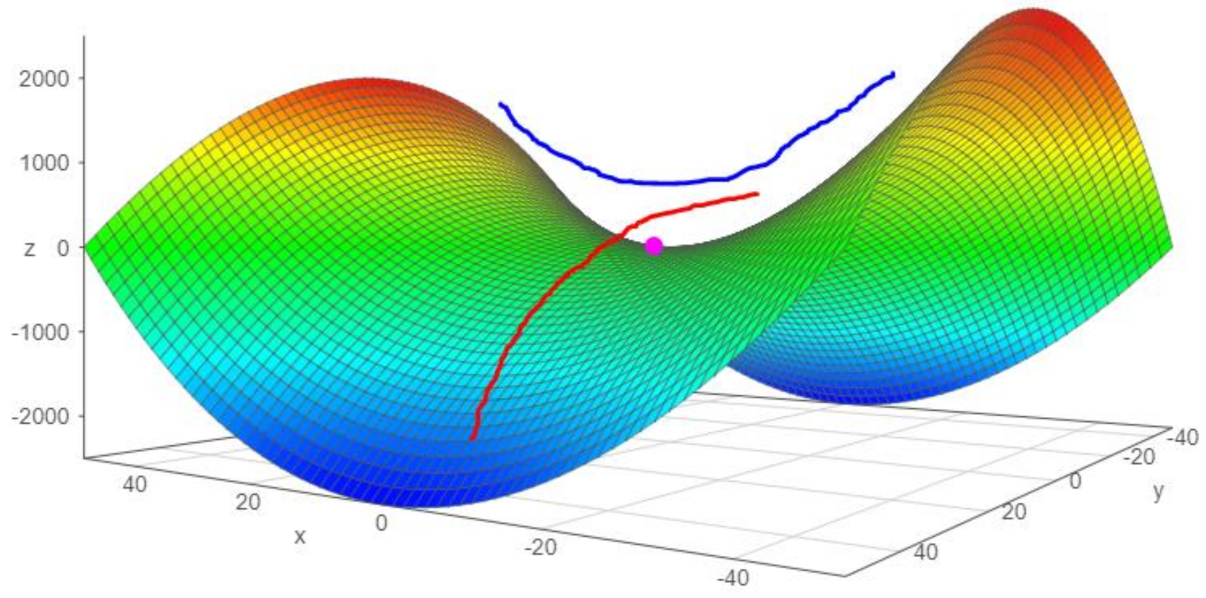


Figure 9: Saddle point

- As discussed previously, choosing a proper learning rate is hard. Also, for mini-batch gradient descent, we have to adjust the learning rate during the training process to make sure it converges to the local minimum and not wander around it. Figuring out the decay rate of the learning rate is also hard and changes with different data sets.
- All parameter updates have the same learning rate; however, we may want to perform larger updates to some parameters that have their directional derivatives more inline with the trajectory towards the minimum than other parameters.

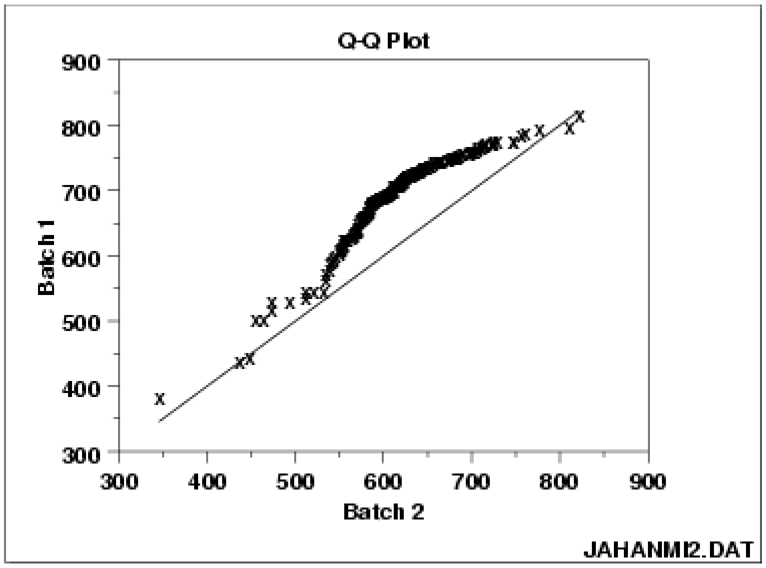
What is a Q-Q plot? Explain the use and importance of a Q-Q plot in linear regression.

The quantile-quantile (q-q) plot is a graphical technique for determining if two data sets come from populations with a common distribution.
A q-q plot is a plot of the quantiles of the first data set against the quantiles of the second data set. By a quantile, we mean the fraction (or percent) of points below the given value. That is, the 0.3 (or 30%) quantile is the point at which 30% percent of the data fall below and 70% fall above that value.

A 45-degree reference line is also plotted. If the two sets come from a population with the same distribution, the points should fall approximately along this reference line. The greater the departure from this reference line, the greater the evidence for the conclusion that the two data sets have come from populations with different distributions.

The advantages of the q-q plot are:

The sample sizes do not need to be equal.
Many distributional aspects can be simultaneously tested. For example, shifts in location, shifts in scale, changes in symmetry, and the presence of outliers can all be detected from this plot. For example, if the two data sets come from populations whose distributions differ only by a shift in location, the points should lie along a straight line that is displaced either up or down from the 45-degree reference line.
The q-q plot is similar to a probability plot. For a probability plot, the quantiles for one of the data samples are replaced with the quantiles of a theoretical distribution.



This q-q plot of the JAHANMI2.DAT data set shows that

These 2 batches do not appear to have come from populations with a common distribution.
The batch 1 values are significantly higher than the corresponding batch 2 values.
The differences are increasing from values 525 to 625. Then the values for the 2 batches get closer again.
The q-q plot is formed by:
Vertical axis: Estimated quantiles from data set 1
Horizontal axis: Estimated quantiles from data set 2
Both axes are in units of their respective data sets. That is, the actual quantile level is not plotted. For a given point on the q-q plot, we know that the quantile level is the same for both points, but not what that quantile level actually is.

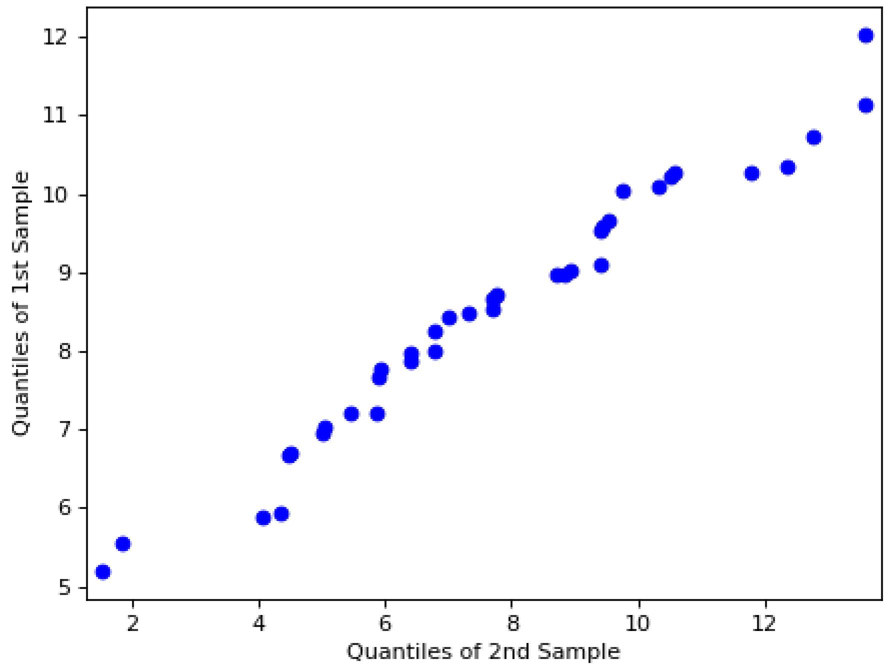
If the data sets have the same size, the q-q plot is essentially a plot of sorted data set 1 against sorted data set 2. If the data sets are not of equal size, the quantiles are usually picked to correspond to the sorted values from the smaller data set and then the quantiles for the larger data set are interpolated.

The q-q plot is used to answer the following questions:

- Do two data sets come from populations with a common distribution?
- Do two data sets have common location and scale?
- Do two data sets have similar distributional shapes?
- Do two data sets have similar tail behavior?

When there are two data samples, it is often desirable to know if the assumption of a common distribution is justified. If so, then location and scale estimators can pool both data sets to obtain estimates of the common location and scale. If two samples do differ, it is also useful to gain some understanding of the differences. The q-q plot can provide more insight into the nature of the difference than analytical methods such as the chi-square and Kolmogorov-Smirnov 2-sample tests.

```
>>> import statsmodels.api as sm
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from statsmodels.graphics.gofplots import qqplot_2samples
>>> x = np.random.normal(loc=8.5, scale=2.5, size=37)
>>> y = np.random.normal(loc=8.0, scale=3.0, size=37)
>>> pp_x = sm.ProbPlot(x)
>>> pp_y = sm.ProbPlot(y)
>>> qqplot_2samples(pp_x, pp_y)
>>> plt.show()
```


Quantiles of 1st Sample
Quantiles of 2nd Sample